1. متغيرات وعمليات في بايثون

نحن نقول إن يمكن الأجهزة الكمبيوتر تخزين البيانات والمعالجة بها باستخدام المتغيرات بيثون هي لغة برمجة عالية المستوى تسمح لمستخدميها بإنشاء هذه المتغيرات ومعالجة البيانات بشكل موجز للغاية مقارنة باللغة البرمجة الأخرى مثل لغة السي ولغة جافا وما إلى ذلك.

هذا الاسبوع سنلقي نظرة على بيثون الأساسي مع تركيز على `CRUD - Create, Read, Update, and Delete` لمعالجة البيانات
في هذا القطاع,سنتعلم كيفية إنشاء المتغيرات وتناول بها لكل أنواع البيانات(eg, string, integer,إلى ألخ)باستخدام بايثون

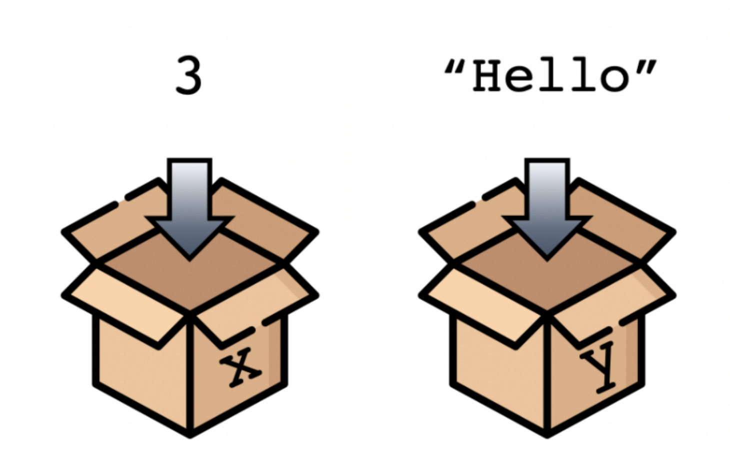
أهداف

1. **متغيرات بايثون:** فهم تعريف وخصائص متغيرات بايثون
2. **أنواع البيانات الأساسية والتشغيل في بايثون:** فهم أنواع وتشغيل بيانات بايثون

1.متغيرات بايثون

1. ما هو متغير؟

- في بايثون المتغير هو ذاكرة محجوزة(يتم تحديدها بواسطة عنوان ذاكرة) مقترنة باسم رمزي مرتبط في تخزين



2.متغيرات في بايثون

- في بايثون يمكن للمتغير تخزين البيانات تحت اسم كود, وستستخدم الكود للوصول إلى البيانات
- يمكن تعيين **قيم(values)** أو **المتغيرات(variables)**و أو **حتى** **نتائج العمليات(results of operations)** إلى المتغير

(1) تعيين قيمة إلى المتغير

في بيثون, يمكنك إنشاء متغير من خلال تخصيص قيمة له باستخدام عامل الإسناد = .
من الممكن أيضًا الكتابة فوق متغير موجود بقيمة جديدة من أي نوع بيانات.

In [1]:
```
a = True # Boolean (true or false)
x = 1 # Integer
y = 1.2 # Float
z = "string" # String

lst = [1,2,3,4,"a","b","c","d"] # List
```

In [2]:
```
# Overwriting x of the value 1 with a new value 'Hello Python'
x = "Hello Python"
```

(2)تسمية الاتفاقيات للمتغيرات

جميع الأسماء المتغيرة **يجب** أن تبدأ بأنجدية أو تحتية(_).
يمكن أن تحتوي الأسماء المتغيرة على أحرف وأرقام ولكن لا يسمح بأي مساحة بيضاء أو أحرف خاصة
.في اسم متغير لا يمكن استخدام الكلمات الرئيسية مثل `true` , `fasle` , `or`
أخيرًا, يجب أن يمثل اسم المتغيرالقيم التي يحملها المتغير

In [3]:
```
# A variable name with letters and numbers
hello1 = "Hello Python"
```

In [4]:
```
# # An exception is a variable name cannot start with a number
# # This code will cause an error.
# 1hello = "Hello Python"
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
-------------------------
File "", line 2
"1hello = "Hello Python
          ^
SyntaxError: invalid syntax
```

يمكنك أيضًا استخدام underscore (_) للدلالة على المساحات البيضاء أو جعل اسم متغير أكثر قابلية للقراءة من قبل الإنسان.
من خلال ، تحتوي underscore أحيانًا على معاني خاصة في سيناريوهات بايثون المختلفة ، وسنناقشها لاحقًا.

```
(Reference: [PEP 8 : Naming Conventions](https://www.python.org/dev/peps/
pep-0008/#naming-conventions))
```

In [5]:
```
# A variable name cannot have whitespaces, but underscores(`_`)
greet_person = "Hello Everyone!"
```

علاوة ذلك ، كل لغة برمجة لديها بعض الكلمات المحجوزة التي لا يتم دعمها كأسماء متغيرة. تسمى هذه `keywords` وتؤدي وظائف محددة مسبقًا.
Reference: [Python Keywords] (https://www.w3schools.com/python/python_ref_keywords.asp)
(((https://www.w3schools.com/python/python_ref_keywords.asp

In [6]:
```
# # 'try' is a Python keyword and returns an error when used as a variable name
# # This code will cause an error.
# try = 1
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
-------------------------
File "", line 1
True = 1
        ^
SyntaxError: cannot assign to True
```

```
In [7]: # # 'True' is a Python keyword and returns an error when used as a variable name
        # # This code will cause an error.
        # True = 1
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ

```
-------------------------
     File "", line 1
        True = 1
           ^
SyntaxError: cannot assign to True
```

(3)تخرين قيمة محسوبة لمتغير

يمكن للمتغيرات تخزين نتائج الحسابات

```
In [8]: # `x` stores 3 which is the outcome of 1 + 2
        x = 1 + 2
        x

Out[8]: 3
```

(4)تعيين متغير إلى متغير

يمكن للمتغير تخزين المتغيرات إلى جانب القيم. عندما يتم تعيين متغير إلى متغير آخر ، يشير المتغيران إلى نفس البيانات (في نفس مساحة الذاكرة) دون إنشاء أي قيمة جديدة. في الأمثلة أدناه،

يمكنك أن ترى أن المتغيرين ، `x` و `y` ، يشيران إلى نفس مساحة الذاكرة عندما يتم تعيين `x` إلى `y` .

```
In [9]: # Create a variable named `x` holding a value and assign'x' to another variable `
        x = 1
        y = x

        print("x : ",x)
        print("y : ",y)

        # x and y share the same memory space
        print("id(x) : ", id(x))
        print("id(y) : ", id(y))

        x :  1
        y :  1
        id(x) :  4373940576
        id(y) :  4373940576
```

(5)حذف المتغيرات

عندما لا تكون هناك حاجة إلى متغير ، يمكنك ببساطة حذفه بكلمة رئيسية `del` كما هو موضح أدناه.

```
In [10]: # # This code will cause an error.
         # del x
         # id(x)
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
                                                                        (t
                                                                       in
                                                            del x 1
                                                            (id(x 2 <----

NameError: name 'x' is not defined
```

بمجرد حذف المتغير ، لن يكون المتغير متاحًا بعد الآن

[ممارسة]:متغيرات في بيثون

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية الكود أدناه

.

تمرين 1- قم بإنشاء متغير `a` وبتعيينه بقيمة 2

In [11]:
```python
# Exercise1 - Answer
a = 2
```

التمرين 2 - قم بإنشاء متغير `b` وبتعيينه قيمة 3.5.

In [12]:
```python
# Exercise2 - Answer
b = 3.5
```

التمرين 3 - قم بإنشاء متغير `c` وبتعيينه قيمة `a` .

In [13]:
```python
# Exercise3 - Answer
c = a
```

تمرين 4 - قم بإنشاء متغير `d` وبتعيينه نتيجة 2 + 3.5.

In [14]:
```python
# Exercise4 - Answer
d = 2 + 3.5
```

تمرين 5- قم بإنشاء متغير `st` وبتعيينه في قيمة `Hello WCO!` .

In [15]:
```python
# Exercise5 - Answer
st = "Hello WCO!"
```

تمرين 6.- قم بإنشاء متغير `li` وبتعيينه ["hello" و 1 و "W" و 2 و "C" و 3 و "O"].

In [16]:
```python
# Exercise6 - Answer
li = [ "hello", 1, 'W', 2, 'C', 3, 'O']
```

[2- أنواع البيانات الأساسية والتشغيل في بايثون]

يدعم بايثون أربعة أنواع بيانات قياسية ، وهي **String** ،**Boolean**، **Float**، **Integer**، ويوفر عددًا من المشغلين للتناول بالبيانات.
نظرًا لأن كل نوع من أنواع البيانات يدعم مجموعة مختلفة من المشغلين ، أو يقوم نفس المشغلين أحيانًا بمهام مختلفة لكل نوع من أنواع البيانات ، فمن المهم معرفة كيفية عمل المشغلين على كل نوع من أنواع البيانات.
ثم ، دعونا نلقي نظرة على أنواع البيانات والمشغلين بمزيد من التفصيل.

integers/float -1

- في بايثون ، هناك نوعان من البيانات للبيانات العددية ، وهما **integers** و **floating-point numbers**.
- يدعم العدد الصحيح والعائم بشكل أساسي المشغلين الحسابيين الأساسيين الأربعة ، `+` ، `-` ، `*` ، `/` على التوالي لإضافة قيم الإضافة والطرح والضرب والقسمة.
- إلى جانب ذلك ، يمكن استخدام المشغلين مثل (floor division)قسم الأرضية (`/`) ، Modulo (`%`) ، وexpolienation (`**`) على integer/float

```
In [17]: # `type()` returns the data type of value inserted

         print(type(1), '1 is an integer, abbreviated as int.')
         print(type(1.2), '1.2 is a floating-point number, abbreviated as float.')

         <class 'int'> 1 is an integer, abbreviated as int.
         <class 'float'> 1.2 is a floating-point number, abbreviated as float.
```

(1)أربع عمليات حسابية قياسية

إضافة

```
In [18]: 1 + 2

Out[18]: 3
```

```
In [19]: # When calculating integer and floating-point values, the result is returned as f
         1 + 1.2

Out[19]: 2.2
```

طرح

```
In [20]: 3 - 2

Out[20]: 1
```

```
In [21]: 3 - 1.4

Out[21]: 1.6
```

ضرب

```
In [22]: 3 * 2

Out[22]: 6
```

```
In [23]: 3.2 * 2

Out[23]: 6.4
```

قسمة

```
In [24]: 4 / 2 # `/` returns a float by default
```

```
Out[24]: 2.0
```

```
In [25]: 4.2 / 2.1
```

```
Out[25]: 2.0
```

```
In [26]: # # This code will cause an error.
         # # Having 0 as a denominator returns ZeroVidisionError
         # 4.2 / 0
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call las
                                                                      (t
                                                                     in
                                                          0 / 4.2 1 <----

                                    ZeroDivisionError: float division by zero
```

(2)مزيد عن المشغلين

Floor division

```
In [27]: 5 // 2 # Returns only the quotient
```

```
Out[27]: 2
```

Modulo

```
In [28]: 5 % 2 # Returns only the remainder
```

```
Out[28]: 1
```

Exponentiation

$$3 * *2 = 3^2$$

```
In [29]: 3 ** 2
```

```
Out[29]: 9
```

```
In [30]: 3 ** 2.1
```

```
Out[30]: 10.04510856630514
```

(3)تعيين صيغة لمتغير

العمليات الحسابية باستخدام المتغيرات
نظرًا ل $x = 3, y = 2$,
يمكنك ببساطة استخدام المتغيرات في العمليات الحسابية مثل $x^2 + y^3 + 2xy + 3x + 4y$.

In [31]:
```
x = 3
y = 2
x**2 + y**3 + 2*x*y + 3*x + 4*y
```
Out[31]: 46

(4) ترتيب العمليات مع بارينسيس ()
كما هو الحال في الرياضيات ، يمكنك استخدام بارينسيس، `()`, لتحديد ترتيب العمليات
نظرًا ل $x = 3, y = 2$,
$x^2 + y^3 + 2xy + 3(x + 4y)$ سيتم حساب $3^2 + 2^3 + 2 * 3 * 2 + 3(3 + 4 * 2)$, نتيجة لذلك 62

In [32]:
```
x = 3
y = 2
x**2 + y**3 + 2*x*y + 3*(x + 4*y)
```
Out[32]: 62

[ممارسة] Integer/Float

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين 7 - قم بإنشاء المتغيرات `p` و `q` ، ثم قم بتعيين 5 و 7 لكل متغير على التوالي.

In [33]:
```
# Exercise7 - Answer
p = 5
q = 7
```

تمرين 8- احسب $p^3 + 3p^2q + 3pq^2 + q^3$ باستخدام المشغلين

In [34]:
```
# Exercise8 - Answer
p**3 + 3*(p**2)*q + 3*p*(q**2) + q**3
```
Out[34]: 1728

تمرين9-احسب $(p + q)^3$ ثم قارن النتيجة مع نتيجة التمرين السابق.

In [35]:
```
# Exercise9 - Answer
(p + q)**3
```
Out[35]: 1728

2 - مقارنات العوامل وBoolean

- في بيثون والبولين هو نوع بيانات يمثل واحدًا فقط من قيمتين منطقيتين محتملتين ، **'True'** أو **'False'**.
- عوامل المقارنة يقارنون بين اثنين من المشغلات ويعيدون قيمة منطقية.

(1) عوامل المقارنة

name	operation
Equal	a == b

operation	name
a != b	Not Equal
a > b	a is greater than b
a >= b	a is greater than or equal to b
a < b	a is less than b
a <= b	a is less than or equal to b

- تقارن عوامل المقارنة بين قيمتين وتعيد إما `True` أو `False` .
- في معظم الحالات ، تؤدي عوامل المقارنة نفس المهام مثل تلك الموجودة في الرياضيات.
- لا يمكن كتابة `!=` و `<=` عكسيًا كـ `=!` أو `=>` .

== (Equal)

In [36]:
```
1 == 1
```
Out[36]:
```
True
```

!= (Not equal)

In [37]:
```
1 != 2
```
Out[37]:
```
True
```

> (Greater)

In [38]:
```
2 > 2
```
Out[38]:
```
False
```

In [39]:
```
'a' <= 'b' # It can also be used on string data
```
Out[39]:
```
True
```

>= (Equal or greater than)

In [40]:
```
2 >= 2
```
Out[40]:
```
True
```

< (Less)

In [41]:
```
2 < 3
```
Out[41]:
```
True
```

<= (Equal or less)

In [42]:
```
2 <= 3
```
Out[42]:
```
True
```

(2) العوامل المنطقية

- في بايثون ، توجد عوامل تشغيل منطقية يمكن استخدامها في عبارتين شرطيتين.

operation	description
a and b	Returns True if both statements are true
a or b	Returns True if one of the statements is true
not b	Reverses the result, returns False if the result is true
a is b	Returns True if both variables are the same object

and

ترجع `and` صواب إذا كانت كلتا العبارتين صحيحة.

In [43]:
```
True and True
```
Out[43]: True

In [44]:
```
True and False
```
Out[44]: False

In [45]:
```
False and False
```
Out[45]: False

or

تقوم بإرجاع `or` صحيح إذا كانت أي من العبارات صحيحة.

In [46]:
```
True or True
```
Out[46]: True

In [47]:
```
True or False
```
Out[47]: True

In [48]:
```
False or False
```
Out[48]: False

not

تُرجع `not` قيمة صحيحة عندما تكون قيمة Boolean خطأ ، والعكس صحيح.

In [49]:
```
not True
```
Out[49]: False

In [50]:
```
not False
```
Out[50]: True

(3) الجمع بين العوامل المنطقية

```python
In [51]: # Given x,y,z as below,
x = 1
y = 3
z = 4

# You can use multiple logical operators in combination with comparison operators

(z > x and z > y) or not (y <= x)
```

```
Out[51]: True
```

عامل `is`

يقارن كل من عامل التشغيل `=` و `is` عاملين. ومع ذلك ، فإن `is` يحدد ما إذا كان هناك عاملان يشيران إلى **نفس مساحة الذاكرة** بينما يتحقق `==` من **المساواة بين قيمتين** .

بمعنى آخر ، `is` يختبر عامل التشغيل هوية المعاملات (ie, العنوان في الذاكرة)

```python
In [52]: # Creating two lists with the same values

a = [1,2,3,4,5]
b = [1,2,3,4,5] # List b containing the same values as list a
```

```python
In [53]: print("a is stored at ", id(a))
print("b is stored at ", id(b))
```

```
a is stored at  140577235878016
b is stored at  140577235950208
```

```python
In [54]: # Testing if values in `a` are equal to those in `b`
a == b
```

```
Out[54]: True
```

```python
In [55]: # Testing if `b` refers to what `a` refers to for id
a is b
```

```
Out[55]: False
```

[ممارسة] مقارنات العوامل وBoolean

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية الكود أدناه.

تمرين 10. تعيين 10 و 4 و 1 للمتغيرات a و b و c على التوالي ، ثم إنشاء تعبير يمكنه التحقق مما إذا كانت البيانات الواردة أدناه صحيحة.

- المتغير a له أكبر قيمة.
- المتغير b له قيمة أكبر من قيمة المتغير c .

```python
In [56]: # Exercise10 - Answer
a = 10
b = 4
c = 1
((a > b) and (a > c)) and (b > c)
```

```
Out[56]: True
```

تمرين 11. اكتب تعبيرًا حيث يتم إرجاع True إذا كانت قيمة المتغير a ليست أكبر من b ، باستخدام عامل التشغيل not .
(تلميح: يجب إرجاع True إذا كانت a أصغر أو تساوي b . يجب إرجاع الخطأ إذا كان a أكبر من b .)

In [57]:
```
# Exercise11 - Answer
not (a > b)
```
Out[57]: False

تمرين 12 .قم بتعيين [1، 2، 3] للمتغير `li1` و [1، 2، 3] للمتغير `li2` . ثم ، قارن ما إذا كان العنوان في الذاكرة متساوًا باستخدام المشغل `is` .

In [58]:
```
# Exercise12 - Answer
li1 = [1, 2, 3]
li2 = [1, 2, 3]
li1 is li2
```
Out[58]: False

تمرين 13. قم بتعيين قيمة المتغير `li1` إلى المتغير `li3` ، ثم استخدام المشغل `is` لمقارنة ما إذا كان العنوان في الذاكرة متساويا أم لا.

In [59]:
```
# Exercise13 - Answer
li3 = li1
li1 is li3
```
Out[59]: True

3. سلاسل(string)

- في بايثون ، السلاسل هي عبارة عن سلسلة من الأحرف. يمكنك استخدام العوامل مباشرة أو وظائف المدمجة على السلاسل
- يوفر بايثون عددًا من الميزات القوية للسلاسل

(1) إنشاء سلاسل

يمكن إنشاء السلاسل عن طريق إرفاق الأحرف داخل الاقتباسات الفردية (`'string'`) أو الاقتباسات المزدوجة (`"string"`).

- مثال: `"Hello "` `'Hello'`

In [60]:
```
str1 = "hello"
print(str1)
```
```
hello
```

إذا كنت تريد سلاسل ووثائق متعددة الأسطر ، فيمكنك استخدام علامات الاقتباس الثلاثية (`strings in multiple lines`) قبل السلاسل وبعدها.

In [61]:
```python
str1 = """
When summer ended
the leaves of snapdragons withered
taking their shrill-colored mouths with them.
They were still, so quiet. They were
violet where umber now is. She hated
and she hated to see
them go. Flowers

born when the weather was good - this
she thinks of, watching...

Emplumada by LORNA DEE CERVANTES
"""
print(str1)
```

```
When summer ended
the leaves of snapdragons withered
taking their shrill-colored mouths with them.
They were still, so quiet. They were
violet where umber now is. She hated
and she hated to see
them go. Flowers

born when the weather was good - this
she thinks of, watching...

Emplumada by LORNA DEE CERVANTES
```

(2)العوامل على السلاسل

يمكنك استخدام `+` or `*` على السلاسل للإضافة والضرب. على الرغم من ذلك ، فإنها تعمل بشكل مختلف عما رأيناه في السابق على الأرقام.

إضافة وضرب السلاسل

In [62]:
```python
# Adding two strings
# If  a `+` is used on strings, it concatenates a string to another
str1 = "Hello, "
str2 = "world!"

str1 + str2
```

Out[62]:
```
'Hello, world!'
```

In [63]:
```python
"""
Multiplying a string
if  a `*` is used on a string, it repeats the string for the number of times
the string is multiplied by
"""

str1 = "Hello!"
str1*3
```

Out[63]:
```
'Hello!Hello!Hello!'
```

(3) `len` - حساب عدد الأحرف

إذا كنت ترغب في معرفة عدد الأحرف التي تتكون منها السلسلة ، فيمكنك بسهولة استخدام وظيفة مدمجة في بيثون `len()` .

In [64]:
```python
str1 = "Hello~ "

len(str1)
```
Out[64]: 7

[ممارسة]سلاسل (strings)

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية الكود أدناه.

تمرين 14. قم بتعيين السلسلة hello ، إلى متغير st1 .

In [65]:
```python
# Exercise14 - Answer
st1 = "Hello,"
```

تمرين 15.قم بتعيين سلسلة WCO! للمتغير st2 .

In [66]:
```python
# Exercise15 - Answer
st2 = "WCO!"
```

تمرين 16. قم بإخراج قيمة المتغير st2 مرتين. (استخدم عامل الضرب بالسلسلة.)

In [67]:
```python
# Exercise16 - Answer
st2*2
```
Out[67]: 'WCO!WCO!'

تمرين 17 قم بتصنيف قيمة السلسلة للمتغير st1 و st2 ، ثم قم بتعيين النتيجة إلى المتغير st3 . أخرج قيمة المتغير st3 .

In [68]:
```python
# Exercise17 - Answer
st3 = st1 + st2
st3
```
Out[68]: 'Hello,WCO!'

تمرين 18. قم بإخراج طول السلسلة المخصصة للمتغير st3 .

In [69]:
```python
# Exercise18 - Answer
len(st3)
```
Out[69]: 10

(4) فهرسة السلاسل

في بايثون ، يتم ترتيب سلسلة من بيانات الأحرف. يسمح لك الفهرسة بالوصول إلى الأحرف الفردية في سلسلة مباشرة باستخدام قيمة موضعية رقمية.
فهرسة السلسلة هي صفر ، مما يعني أن العنصر الأول من السلسلة هو 0 ثم 1 وهكذا دواليك. يمكنك الوصول إلى كل حرف بالإشارة إلى قيمته الموضعية بين قوسين مربعين أو أقواس فهرسة [ ] .
هنا ، لاحظ أنه يمكنك أيضًا تحديد مجموعة من الأحرف من سلسلة عن طريق إنشاء مجموعة من أرقام الفهرس مفصولة بقولون [x: y]. الرقم القياسي الأول x هو المكان الذي تبدأ فيه الشريحة (شاملة) ، ورقم الفهرس الثاني y هو المكان الذي تنتهي فيه الشريحة. (حصري)

In [70]:
```python
# String indexing
str1 = "This is Python World!"
```

```
In [71]: # Accessing the 4th value
         str1[3]
```

```
Out[71]: 's'
```

```
In [72]: # Extracting values from the 0th position to less than the 4th
         str1[0:4]
```

```
Out[72]: 'This'
```

(5) تقسيم سلسلة

يمكنك تقسيم سلسلة باستخدام وظيفة القسمة ( ) .

يقسم سلسلة إلى قائمة حيث كل كلمة عنصر قائمة. يمكنك تحديد الفاصل. بشكل افتراضي ، الفاصل هو أي مساحة بيضاء.

split() syntax: `.split(separator, maxsplit)`

```
In [73]: str1 = "This is Python World!"
```

```
In [74]: # # Splitting a string on any whitespaces
         # # This code will cause an error.
         # str1.split(" ")
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
                                                                       (t
                                                                      in
                         Splitting a string on any whitespaces # 1
                                                 (" ")str1.split 2 <----

                                  NameError: name 'str1' is not defined
```

```
In [75]: # Splitting a string on any 's'
         str1.split('s')
```

```
Out[75]: ['Thi', ' i', ' Python World!']
```

```
In [76]: str1.split(' ', 2) # Splitting the string only twice
```

```
Out[76]: ['This', 'is', 'Python World!']
```

(6) تنسيق السلسلة(string formatting)

يستخدم تنسيق السلسلة عملية استيفاء السلسلة لتقييم سلسلة حرفي تحتوي على واحد أو أكثر من أصحاب النواة ، مما يؤدي إلى نتيجة يتم فيها استبدال أصحاب النواة بقيمهم المقابلة.

على سبيل المثال ، هناك جمل تحتاج إلى تكرارها في كل حقبة في التعلم العميق.

```
"val_loss_change = "Epoch 00001: val_loss improved from inf to 0.15311
```

إذا كنت ترغب في الاحتفاظ فقط بتنسيق هذه الجملة بينما تتغير القيمة مرارًا وتكرارًا ، فيمكنك القيام بشيء كهذا.

```
In [77]: step = 32
         percent = 92.3

         info = "Epoch " + str(step) +": val_loss improved from inf to "+ str(percent)
         info
Out[77]: 'Epoch 32: val_loss improved from inf to 92.3'
```

تنسيق السلسلة بـ `.format`

```
In [78]: "Epoch {}: val_loss improved from inf to {}".format(step, percent)
Out[78]: 'Epoch 32: val_loss improved from inf to 92.3'
```

تنسيق السلسلة بـ `f-string`

```
In [79]: f"Epoch {step}: val_loss improved from inf to {percent} "
Out[79]: 'Epoch 32: val_loss improved from inf to 92.3 '
```

[ممارسة]سلاسل2 (strings2)

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية الكود أدناه.

تمرين 19. قم بتعيين سلسلة `WCO is World Customs Organization` للمتغير `st4` .

```
In [80]: # Exercise19 - Answer
         st4 = "WCO is World Customs Organization"
```

تمرين 20 باستخدام فهرسة السلسلة ، قم بأخراج فقط سلسلة `World Customs Organazation` من المتغير `st4` .

```
In [81]: # Exercise20 - Answer
         st4[7:]
Out[81]: 'World Customs Organization'
```

تمرين21.قم بتقسيم السلسلة المخزنة في المتغير `st4` ، باستخدام المسافة البيضاء كفاصل. إخراج النتيجة.

```
In [82]: # Exercise21 - Answer
         st4.split(' ')
Out[82]: ['WCO', 'is', 'World', 'Customs', 'Organization']
```

تمرين 22. قم بتعيين 178 إلى المتغير `num` . الاحتفاظ بصيغة `There are member states participating in the WCO` ، استبدال بقيمة `num` . قم بإخراج النتيجة المنسقة.

```
In [83]: # Exercise22 - Answer
         num = 178
         "There are {} member states participating in the WCO.".format(num)
         f"There are {num} member states participating in the WCO."
Out[83]: 'There are 178 member states participating in the WCO.'
```

2. هياكل البيانات

في الدرس السابق ، قيل إن الكمبيوتر يمكنها تخزين ومعالجة العديد من أنواع البيانات. وبما أن كمية البيانات التي تكون عمليات الكمبيوتر ضخمة ، يجب تخزين البيانات بكفاءة للوصول والعملية في الوقت المناسب. من أجل تحقيق ذلك ، نستخدم شيئًا يسمى "بنيات البيانات". لذا ، دعونا ندرس من خلال الموضوعات التي يجب أن نغطيها حول هياكل بيانات بايثون

الموضوعات

1.هياكل البيانات

: فهم كل نوع من هياكل البيانات وتعلم كيفية التعامل معها

Data structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 5, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 5, 'yes', 5.7)`
Set	No	Yes	`{ }` or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes	`{'key' : values }` or `dict()`	`{'Jun' : 75, 'Jul' : 89}`

1.قائمة List [ ]

قائمة هي واحدة من هياكل البيانات التي هي سلسلة قابلة للتغيير أو قابلة للتغيير وأمر من العناصر

1.السمات والمتغيرات في بايثون

في بايثون يتم إنشاء قائمة عن طريق وضع جميع العناصر داخل الأقواس المربعة `` `[]` `` , مفصولة بالفواصل

Listعبارة عن تسلسل مرتبة للعناصر مع كل عنصر مرتبط برقم موضعي يسمى `'index'` تستخدم بايثون وظيفة الفهرسة والمؤشرات الصفرية كمؤشرات تستخدم للوصول إلى عناصر محددة.

علاوة على ذلك ، يوفر بايثون عددًا من الوظائف والطرق المدمجة للتلاعب بالقائمة. هي كذلك `index, append, insert,` `extend, remove,pop` , إلى ألخ

(1) انشاء قائمة

في بايثون ، يمكنك إنشاء بقائمة `` `[]` `` أو `` `قائمة` `` `'()'` . يمكن للقائمة الاحتفاظ بقيم أي نوع من البيانات مع كل قيمة مفصولة بفواصل.

In [1]:
```python
lst0 = [] # Creating an empty list
lst0 = list()
lst0
```
Out[1]:
```
[]
```

In [2]:
```python
lst1 = [1,2,3,4,5] # Creating a list of multiple values
lst1
```
Out[2]:
```
[1, 2, 3, 4, 5]
```

In [3]:
```python
lst2 = [1,2,3,5.5,"str1"] # List of a string and 5 integers
lst2
```
Out[3]:
```
[1, 2, 3, 5.5, 'str1']
```

In [4]:
```python
var1 = 1 # Creating a list containing variables and numbers
lst3 = [var1, 2,3,4, lst2]
lst3
```
Out[4]:
```
[1, 2, 3, 4, [1, 2, 3, 5.5, 'str1']]
```

(2) فهرسة 'قائمة'

قائمة عبارة عن تسلسل أمر للعناصر ، ويرتبط كل عنصر بفهرس. يمكنك تحديد أو الوصول إلى كل قيمة عنصر عن طريق الإشارة إلى فهرسها في [ ].
ملاحظة أن بايثون تستخدم فهرسة تستند إلى الصفر ، حيث يرتبط العنصر الأول بقيمة الفهرسة 0. بناء جملة فهرسة القائمة هو نفسه مثل الفهرسة tuple وسلاسل.

In [5]:
```python
lst = [1, 2, 3, 4, 5]
```

In [6]:
```python
# Accessing the element of the index value of 2 in Ist
lst[2]
```

Out[6]: 3

In [7]:
```python
"""
Python supports negative indexing proceeding backwards
from the last element to the first element of a list
"""

"""
In negative indexing, the first index value -1 is linked
to the last element of the list, -2 to the second last element and so on
"""
lst[-1]
```

Out[7]: 5

In [8]:
```python
"""
Selecting a range of elements in a list using a colon (:)
Accessing the list elements of index 1(inclusive) to 4 (exclusive)
the number on the left indicates the starting index (inclusive)
and that on the right is where the access ceases
"""
lst[1:4]
```

Out[8]: [2, 3, 4]

In [9]:
```python
# Accessing the list starting at index 3 to the end
lst[3:]
```

Out[9]: [4, 5]

In [10]:
```python
"""
You can insert an argument for "step" to determine the amount
by which the index increases, defaults to 1

If `step` is a negative number, the list slicing proceeds
in reverse order
"""
lst[0:5:2] # Accessing every second element of lst
```

Out[10]: [1, 3, 5]

In [11]:
```python
# `in` operator checks if the given value exists in the list
1 in lst
```

Out[11]: True

(2)تعديل القائمة

قائمة بايثون قابلة للتغيير ، والتي تقول أنه يمكن تعديل العناصر باستخدام قيم الفهرسة.

```
In [12]: # Changing the third element of lst to 6
         lst = [1,2,3,4,5]
         lst[2] = 6
         lst
```

```
Out[12]: [1, 2, 6, 4, 5]
```

إذا كانت القائمة طويلة جدًا ، فسيكون من الصعب العثور على فهرس العناصر التي نريد الوصول إليها. في هذه الحالة ، يمكنك استخدام `index` للعثور على مؤشر قيمة الفائدة عند حدوثه الأول.

```
In [13]: lst = [1,2,3,4,5]
         lst.index(4) # Searching for the index where a value 4 appears for the first time
```

```
Out[13]: 3
```

```
In [14]: # #  This code will cause an error.
         # lst.index(7)
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

(4) إضافة عناصر إلى قائمة

في بايثون ، هناك أوقات تحتاج إلى إضافة عناصر جديدة إلى مكان ما في قائمة ، إلى النهاية ، أو البداية ، أو إلى الوسط. في هذا الصدد ، توفر بايثون ثلاث طرق مختلفة ، وهي `append` و`extend` و `insert`. دعونا ننظر في كيفية تنفيذ كل منها

```
In [15]: lst = [1,2,3,4,5]
```

. `Append` - إضافة حجته كعنصر واحد إلى نهاية القائمة. يزيد طول (حجم) القائمة بمقدار واحد.

```
In [16]: # Adding a new value to the end of a list
         lst.append(6)
         lst
```

```
Out[16]: [1, 2, 3, 4, 5, 6]
```

```
In [17]: # Adding a new list to the end of a list
         lst.append([1,2])
         lst
```

```
Out[17]: [1, 2, 3, 4, 5, 6, [1, 2]]
```

`extend` -تكرار على حسب جاجته ويضيف كل عنصر إلى القائمة ويوسع القائمة. يزيد طول القائمة بعدد العناصر في حسب الحاجة.

```
In [18]: # Adding two values to the end of lst
         lst.extend([1,2])
         lst
```

```
Out[18]: [1, 2, 3, 4, 5, 6, [1, 2], 1, 2]
```

`insert` - يدرج عناصر جديدة في موضع محدد داخل القائمة المحددة. يزيد طول القائمة بمقدار واحد.

```
In [19]: # Inserting a value to the specific position of the list
         """
         In this case, we're inserting a float, 2.5,
         to the position of index 2 of the list.
         """
         lst.insert(2,2.5)
         lst.insert(2,2.5)
         lst

Out[19]: [1, 2, 2.5, 2.5, 3, 4, 5, 6, [1, 2], 1, 2]
```

(5) حذف العناصر من القائمة

يمكنك حذف عناصر القائمة باستخدام `del, remove` أو `pop`.

`del` - إزالة العنصر في فهرس محدد.

```
In [20]: lst = [1,2,3,4,5]
         del lst[0] # Removing by referring to the index value
         lst

Out[20]: [2, 3, 4, 5]
```

`remove` - إزالة قيمة المطابقة الأولى ، وليس فهرسًا محددًا.

```
In [21]: lst = [1,2,3,4,5]
         lst.remove(4) # Removing an element by referring to the value to be removed
         lst

Out[21]: [1, 2, 3, 5]

In [22]: # # This code will cause an error.
         # lst.remove(0)
         # lst
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call las
                                                                        (t
                                                                        in

                                                   (lst.remove(0 1 <----
                                                                  lst 2

                                          ValueError: list.remove(x): x not in list
```

`pop` - إزالة العنصر في فهرس معين ويعيد القيمة المحذوفة.

```
In [23]: lst.pop(2) # Returns the element of the index to be removed

Out[23]: 3
```

```
In [24]: lst
```

```
Out[24]: [1, 2, 5]
```

[ممارسة] قائمة الخصائص والعمليات

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين1. إنشاء قائمة فارغة ، ثم تعيينها لمتغير `num_list` .

```
In [25]: # Exercise1 - Answer
num_list = []
```

تمرين2. حفظ [1 ، 2 ، "A"] لمتغير `num_list` .

```
In [26]: # Exercise2 - Answer
num_list = [1, 2, "A"]
```

تمرين3.قم بإخراج "A" من `num_list` , باستخدام الفهرسة

```
In [27]: # Exercise3 - Answer
num_list[2]
```

```
Out[27]: 'A'
```

تمرين4. قم بإضافة 12 و 5 إلى نهاية `num_list` , بالترتيب.

```
In [28]: # Exercise4 - Answer
num_list.extend([12, 5])
```

تمرين5. حذف "A" من `num_list` .

```
In [29]: # Exercise5 - Answer
del num_list[2]
```

تمرين6. قم بتحديث من 12 إلى 3 في `num_list` .

```
In [30]: # Exercise6 - Answer
num_list[2] = 3
```

تمرين7. أضف 4 إلى المركز الثالث `num_list` .

```
In [31]: # Exercise7 - Answer
num_list.insert(3, 4)
```

تمرين8. قم بإخراج `num_list` .

```
In [32]: # Exercise8 - Answer
         num_list
```

```
Out[32]: [1, 2, 3, 4, 5]
```

2. عمليات ووظائف القائمة

يمكنك استخدام العمليات في القوائم تمامًا كما هو الحال في السلاسل أو الأعداد الصحيحة.
`+` و `*` عمليات تمدد قائمة و `zip` يجمع عناصر من قائمتين.
يمكنك أيضًا استخدام وظائف أخرى مدمجة مثل `len` لقياس طول القائمة أو `sorted` لفرز عناصر القائمة.

(1) الجمع(+)

+ العملية تقوم بإنشاء قائمة جديدة عن طريق تصنيف القوائم المحددة

```
In [33]: lst1 = [1,2,3,4,5]
         lst2 = [6,7,8,9,10]
         lst1 + lst2
```

```
Out[33]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

(2) الضرب (*)

* العملية تقوم بإنشاء قائمة جديدة عن طريق تكرار القائمة المحددة لعدد المرات المحدد.

```
In [34]: lst1 * 3
```

```
Out[34]: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

(3) الانضمام إلى قائمتين - zip

`zip` وظيفة تقوم بإرجاع موجود zip ، وهو عبارة عن تكرار حيث يتم إقران العناصر الأولى من القوائم المحددة معًا ، ثم العناصر الثانية ، وما إلى ذلك.

```
In [35]: lst1 = [1,2,3,4,5]
         lst2 = ["a","b","c","d","e"]
         list(zip(lst1,lst2))
```

```
Out[35]: [(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

(4) التحقق من طول القائمة - len

`len` يعيد عدد العناصر في القائمة .

```
In [36]: lst = [1,2,3,4,5]
         len(lst)
```

```
Out[36]: 5
```

(5) فرز عناصر القائمة-,sorted

`sorted` يقوم بفرز عناصر القائمة المحددة بترتيب محدد ، عادة إما بترتيب تصاعدي أو تنازلي.

```
In [37]: lst = [1,3,2,4,5]
```

```
In [38]: # Sorting list elements in ascending order (Default)
         sorted(lst)
```

```
Out[38]: [1, 2, 3, 4, 5]
```

In [39]:
```
# Sorting list elements in descending order
sorted(lst,reverse=True)
```
Out[39]:
```
[5, 4, 3, 2, 1]
```

[ممارسة] مشغلين ووظائف القائمة

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين9. حفظ [1, 3, 5] للتمتغيرات `list1` .

In [40]:
```
# Exercise9 - Answer
list1 = [1, 3, 5]
```

تمرين10. حفظ [2, 4, 6] للمتغيرات `list2`

In [41]:
```
# Exercise10 - Answer
list2 = [2, 4, 6]
```

تمرين11. قم بانضمام `list1` و , `list2` . تخزين النتيجة إلى `list3` .

In [42]:
```
# Exercise11 - Answer
list3 = list1 + list2
```

تمرين12.قم بإخراج طول القائمة `list3` .

In [43]:
```
# Exercise12 - Answer
len(list3)
```
Out[43]:
```
6
```

تمرين 13. فرز `list3` في ترتيب تنازلي.

In [44]:
```
# Exercise13 - Answer
sorted(list3, reverse=True)
```
Out[44]:
```
[6, 5, 4, 3, 2, 1]
```

تمرين14. حفظ ["x", "y", "z"] للمتغيرات `list4` .

In [45]:
```
# Exercise14 - Answer
list4 = ["x", "y", "z"]
```

تمرين15. استخدام `list1` و `list4` ,إخراج النتيجة التالية:

- **[(1, 'x'), (3, 'y'), (5, 'z')]**

In [46]:
```
# Exercise15 - Answer
list(zip(list1, list4))
```
Out[46]:
```
[(1, 'x'), (3, 'y'), (5, 'z')]
```

[2. تعديد]
*تعديد:tuple

تعديد هو هيكل البيانات يحتوي على مجموعة مرتبة من العناصر المنفصلة عن الفاصلة تمامًا كما تسرد بايثون. ولكن هذا **ثابت (غير قابل للتغيير)** وبالتالي أقل مرونة من القوائم.

1. الخصائص والعمليات

- يتم إنشاؤه عن طريق وضع العناصر بين في تعديد `tuple` .
- تعديد هو سلسلة من القيم مع قيم الفهرسة

(1)خلق تعديد

In [47]:
```
tpl = () # Creating an empty tuple
tpl = tuple()
tpl
```
Out[47]:
```
()
```

In [48]:
```
tpl = (1,2,3,4,5) # Creating a tuple of five elements
tpl
```
Out[48]:
```
(1, 2, 3, 4, 5)
```

In [49]:
```
tpl = (1,2,3,5.5,"str1") # Creating a tuple of five numbers and a string
tpl
```
Out[49]:
```
(1, 2, 3, 5.5, 'str1')
```

In [50]:
```
var_1 = 1
tpl_1 = (var_1, 2,3,4, tpl) # Creating a tuple of three numbers and two variables
tpl_1
```
Out[50]:
```
(1, 2, 3, 4, (1, 2, 3, 5.5, 'str1'))
```

(2) فهرسة التعديد
بناء الجملة من فهرسة التوبل هو نفسه من القائمة والسلاسل.

(3)تعديل التعديد غير مدعوم
إن التعديد عبارة عن هيكل البيانات غير قابلة للتغيير ولا تسمح بأي تغيير في محتواها بمجرد إنشائها. ستؤدي محاولة لأي تعديل إلى إثارة خطأ كما تراه في المثال أدناه.

In [51]:
```
# # This code will cause an error.
# tpl= (1,2,3,4,5)
# tpl[2] = 6
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
                                                                        (t
                                                                       in
                                            (tpl= (1,2,3,4,5 3
                                                  tpl[2] = 6 4 <----

TypeError: 'tuple' object does not support item assignment
```

2. مشغلين وعمليات التعديد

- يعمل المشغلون على التعديد بنفس الطريقة الموجودة في القوائم.

[ممارسة] تعديد

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين16. حفظ(1, 3) لمتغير `tuple1` .

In [52]:
```python
# Exercise16 - Answer
tuple1 = (1, 3)
```

تمرين17. حفظ (2, 4)لمتغير `tuple2` .

In [53]:
```python
# Exercise17 - Answer
tuple2 = (2, 4)
```

تمرين18. قم بانضمام `tuple1` و `tuple2` . تخزين النتيجة إلى متغير `tuple3` .

In [54]:
```python
# Exercise18 - Answer
tuple3 = tuple1 + tuple2
```

تمرين19. قم بإخراج قائمة الطول `tuple3` .

In [55]:
```python
# Exercise19 - Answer
len(tuple3)
```

Out[55]: 4

تمرين20. قم بفرز `tuple3` في ترتيب تصاعدي.

In [56]:
```python
# Exercise20 - Answer
sorted(tuple3)
```

Out[56]: [1, 2, 3, 4]

تمرين21. استخدم `tuple1` و `tuple2`, قم بإخراج النتيجة التالية:

- **((1, 2), (3, 4))**

```
In [57]: # Exercise21 - Answer
         tuple(zip(tuple1, tuple2))
```

```
Out[57]: ((1, 2), (3, 4))
```

[Set .3]

set هو واحد من هيكل بيانات بايثون الذي يحتوي على مجموعة غير مرتبة من القيم الفريدة. كمجموعة من القيم الفريدة ، لا تقبل المجموعة أي نسخ مكررة.

1. الخصائص والعمليات

- يتم استخدام مجموعة كوسيلة لتجنب تكرار البيانات لأنها لا تسمح بالتكرار.
- كونها مجموعة غير مرتبة من القيم الفريدة ، فإن العناصر set غير مفهرسة و غير قادرة على الوصول إليها بالطريقة التي نصل بها إلى قائمة أو عناصر التعديد.

1) إنشاء set

'Set' يتم إنشاؤه عن طريق التفاف القيم الفريدة المنفصلة عن الفاصلة مع الأقواس المجعدة `{ }` أو من خلال تدوين المجموعة `set`. يعيّن قيم قبول أو في أي نوع أو بنية بيانات باستثناء قائمة بايثون

```
In [58]: set0 = set() # Creating an empty set
         set0
```

```
Out[58]: set()
```

```
In [59]: set1 = {1, 1, 2, "2", "text", "text", 3} # Creating a set for unique integers and
         set1
```

```
Out[59]: {1, 2, '2', 3, 'text'}
```

```
In [60]: # # This code will cause an error.
         # # Sets do not accept list objects
         # set2 = {1, 2, [1, 2, 3]}
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
 ---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
                                                                        (t
                                                                       in
                                          Sets do not accept list objects # 1
                                                 {[set2 = {1, 2, [1, 2, 3 2 <----

                                               'TypeError: unhashable type: 'list
```

```
In [61]: set2 = {1, 2, (1,2,3)} # The order of tuple elements is different from the order
         set2
```

```
Out[61]: {(1, 2, 3), 1, 2}
```

(2) الوصول إلى عناصر محددة

عناصر المجموعة غير مفهرسة ، ولا يتم دعم بنية الفهرسة التقليدية على المجموعات. بدلاً من ذلك ، يمكنك التحقق مما إذا كانت هناك قيمة محددة في مجموعة باستخدام مشغل "in".

In [62]:
```
# # This code will cause an error.
# # Set indexing available? No. set elements are not indexed
# set1 = {1, 1, 2, "2", "text", "text", 3}
# set1[0]
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call las
                                                                       (t
                                                                      in
       Set indexing available? No. set elements are not indexed # 1
                        {set1 = {1, 1, 2, "2", "text", "text", 3 2
                                                       [set1[0 3 <----

                              TypeError: 'set' object is not subscriptable
```

In [63]:
```
set1 = {1, 2, 3, 4, 5} # Checking the existence of a specific value
print('6 in set1?', 6 in set1)
print('1 in set1?', 1 in set1)
```

```
6 in set1? False
1 in set1? True
```

(3) إضافة عناصر جديدة إلى مجموعة

يمكنك إضافة عناصر جديدة باستخدام `add` أو `update` اعتمادا على عدد العناصر المراد إضافتها.

In [64]:
```
set1 = {1, 2, 3, 4, 5}
```

In [65]:
```
set1.add(6) # Adding a single element to an existing set
set1
```

Out[65]: `{1, 2, 3, 4, 5, 6}`

In [66]:
```
set1.update({7,8,9}) # Adding two or more elements to an existing set
set1
```

Out[66]: `{1, 2, 3, 4, 5, 6, 7, 8, 9}`

(4) حذف العناصر من مجموعة

يمكن حذف عناصر المجموعة باستخدام `remove`

In [67]:
```
set1 = {1,2,3,4,5}
```

In [68]:
```
set1.remove(4) # Removing 4 from set1
set1
```

Out[68]: `{1, 2, 3, 5}`

2. وظائف وعمليات Set

- المجموعة ، كما يوحي الاسم ، تدعم عمليات المجموعة التالي Set

Operation	Description
&	Intersection - elements that exist in both set
|	Union - elements from both sets excluding duplicates
$A-B$	Difference - elements in A that are not in B
^	symmetric difference - elements in either A or B, but not in their intersection

In [69]:
```
A = {1, 2, 3, 5}
B = {3, 5, 6, 8, 9}
```

(1) التقاطع

- تُرجع مجموعة من القيم الفريدة الموجودة في set A **and** set B.

In [70]:
```
A & B
```
Out[70]:
```
{3, 5}
```

(2) الوحدة

- لعرض مجموعة من القيم الفريدة التي هي إما في set A **or** set B, بما في ذلك intersection.

In [71]:
```
A | B
```
Out[71]:
```
{1, 2, 3, 5, 6, 8, 9}
```

(3) الفرق

- تُرجع مجموعة من القيم الفريدة الموجودة في set A but not in set B,.

In [72]:
```
A - B
```
Out[72]:
```
{1, 2}
```

(4) فرق متماثل

- تقوم بإرجاع مجموعة من القيم الفريدة الموجودة في أي من set A or set B,باستثناء المقاطع

In [73]:
```
A ^ B
```
Out[73]:
```
{1, 2, 6, 8, 9}
```

In [74]:
```
set1 = {1,2,3,4,5}
len(set1)
```
Out[74]:
```
5
```

(5) إيجاد طول المجموعة - `len`

`len` تُرجع عدد العناصر في مجموعة.

In [75]:
```
set1 = {1,2,3,4,5}
len(set1)
```
Out[75]:
```
5
```

(6) فرز عناصر المجموعة - `sorted`

على الرغم من أن المجموعة عبارة عن مجموعة غير مرتبة وغير مفهرسة من القيم الفريدة ، يمكن فرز عناصر المجموعة باستخدام sorted ,
والتي تقوم بعد ذلك بإرجاع قائمة مرتبة للمجموعة المحددة بترتيب معين.

In [76]:
```
set1 = {5, 3, 1, 2, 4}
sorted(set1)
```

Out[76]: [1, 2, 3, 4, 5]

[ممارسة] Set

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين22. حفظ{1, 3, 4} لمتغير set1 .

In [77]:
```
# Exercise22 - Answer
set1 = {1, 3, 4}
```

تمرين23. حفظ{2, 3, 5}لمتغير set2 .

In [78]:
```
# Exercise23 - Answer
set2 = {2, 3, 5}
```

تمرين24. اكتب تعبيرًا للتحقق مما إذا كانت المجموعة set1 تشمل 3 و 6.

In [79]:
```
# Exercise24 - Answer
3 in set1
6 in set1
```

Out[79]: False

تمرين25. أضف 6, 7 إلى مجموعة set1 .

In [80]:
```
# Exercise25 - Answer
set1.update({6, 7})
```

تمرين26. قم بإخراج تقاطع set2 and set1 .

In [81]:
```
# Exercise26 - Answer
set1 & set2
```

Out[81]: {3}

تمرين27. قم بإخراج التوحيد set1 و set2 .

In [82]:
```
# Exercise27 - Answer
set1 | set2
```

Out[82]: {1, 2, 3, 4, 5, 6, 7}

تمرين28. استخدم set1 و set2 ,حساب وقم بإخراج النتيجة التالية

- **{1, 2, 4, 5, 6, 7}**

In [83]:
```
# Exercise28 - Answer
set1 ^ set2
```
Out[83]:
```
{1, 2, 4, 5, 6, 7}
```

4]. قاموس :dictionary]

القاموس (اختصار Dict) هو أحد هياكل بيانات بايثون التي تخزن أزواج **مفتاح / قيمة** .

1. خلق القاموس

- تم إنشاء القاموس باستخدام `{'key':'value'}` أو `dict`.
- القاموس هو مجموعة غير مرتبة من أزواج المفتاح والقيمة. يعمل مفتاح القاموس كمعرف للقيمة المرتبطة ، ويمكن البحث عن قيمة القاموس بواسطة مفتاحه.
- يمكن للقاموس تخزين البيانات من أي نوع.
- لا يقبل القاموس التكرارات لمفاتيحه.
- يمكن أن تكون المفاتيح في القاموس من أي **نوع بيانات**.

(1) خلق القاموس

يحتوي القاموس على **مفتاح/قيمة** أزواج ويمكن إنشاؤها بواسطة `{'مفتاح': 'قيمة'}` أو `dict({'مفتاح': 'قيمة'})`

In [84]:
```
dict0 = {} # Creating an empty dictionary
dict0 = dict()
dict0
```
Out[84]:
```
{}
```

In [85]:
```
dict1 = dict({"1": 2})
dict1
```
Out[85]:
```
{'1': 2}
```

In [86]:
```
dict1 = {'1': 'a',
        '2':'b',
        '3': 1,
        'c': 'test'} # Creating a dictionary of four key/value pairs
dict1
```
Out[86]:
```
{'1': 'a', '2': 'b', '3': 1, 'c': 'test'}
```

In [87]:
```
dict1 = {'1': 'a',
        '2': [1,2,3,4],
        '3': (1,2,3),
        'c': {1:'1',2:'2'}} # A dictionary accepts other data structures such as
dict1
```
Out[87]:
```
{'1': 'a', '2': [1, 2, 3, 4], '3': (1, 2, 3), 'c': {1: '1', 2: '2'}}
```

(2) فهرسة القاموس

ترتبط القيم الموجودة في القاموس بقيم المفاتيح بدلاً من الفهرس ويمكن الوصول إليها من خلال مفتاحه.

In [88]:
```
dict1 = {"1":"a",
         "2":"b",
         "3": 4}
```

In [89]:
```
dict1['1'] # Searching for a value of the key "1"
```
Out[89]:
```
'a'
```

In [90]:
```
# # This code will cause an error.
# dict1[3] # Strings and integers are to be distinguished
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call las
                                                                         (t
                                                                        in
                                                     [dict1[3 1 <----

                                                              KeyError: 3
```

باستخدام `in` ، يمكنك التحقق مما إذا كانت قيمة المفتاح المحددة موجودة أم لا في القاموس.

In [91]:
```
"1" in dict1
```

Out[91]: True

In [92]:
```
3 in dict1
```

Out[92]: False

إذا كانت هناك قائمة بالقيم في القاموس ، فيمكن الوصول إلى العنصر الموجود في موضع معين في القائمة عن طريق الفهرس.

In [93]:
```
dict1 = {'1':'a',
         '2':[1,2,3,4],
         '3':(1,2,3),
         'c':{1:'1',2:'2'}}
dict1['2'][0] # Getting the value at index 0 from the list whose key value is '2
```

Out[93]: 1

(3) تعديل القاموس

In [94]:
```
dict2 = {"1":"a","2":"b","3":"4"}
```

In [95]:
```
dict2['4'] = 'hello' # Adding a value "hello" of a key '4' in dict2
dict2
```

Out[95]: {'1': 'a', '2': 'b', '3': '4', '4': 'hello'}

In [96]:
```
dict2['1'] = 10 # Changing the value of the key '1' from 'a' to 10
dict2
```

Out[96]: {'1': 10, '2': 'b', '3': '4', '4': 'hello'}

(4) حذف عناصر القاموس

يمكنك حذف القيمة في القاموس باستخدام `del` .

In [97]:
```
dict2 = {"1":"a","2":"b","3":"4"}
```

```
In [98]: del dict2['1'] # Using `del` to remove a key/value pair of the key value '1'
         dict2
```
```
Out[98]: {'2': 'b', '3': '4'}
```

2. عوامل تشغيل القاموس والوظائف

- موجودات قاموس بايثون لا تدعم +و * و zip
- len تحسب عدد أزواج المفاتيح / القيم في القاموس (الطول) ، و sorted تفرز القيم الأساسية بترتيب معين.

(1) التحقق من الحجم مع len

- len ترجع كم **مفاتيح** في القاموس

```
In [99]: dict1 = {1:2,2:[3,4,5]}
         len(dict1)
```
```
Out[99]: 2
```

(2) فرز بواسط sorted

- sorted يفرز القاموس **مفاتيح** فقط ، إرجاع قائمة لقيمة المفاتيح التي تم فرزها.

```
In [100]: dict1 = {1:2,2:[3,4,5]}
          sorted(dict1,reverse=True)
```
```
Out[100]: [2, 1]
```

[ممارسة] قاموس

استنادًا إلى ما تعلمته ، اكتب الإجابات عن الأسئلة الموجودة في خلية التعليمات البرمجية أدناه.

تمرين29.حفظ {John':33, 'Lucy':28, 'Lee':29, 'Chris':188'} إلى متغير age_dict .

```
In [101]: # Exercise29 - Answer
          age_dict = {'John':33,
                      'Lucy':28,
                      'Lee':29,
                      'Chris':188}
```

تمرين30. قم بإخراج قيمة Lucy من المتغير age_dict .

```
In [102]: # Exercise30 - Answer
          age_dict['Lucy']
```
```
Out[102]: 28
```

تمرين31.قم بتحديث القيمة Lee من المتغير age_dict إلى 28.

```
In [103]: # Exercise31 - Answer
          age_dict['Lee'] = 28
```

تمرين32.احذف مفتاح وقيمة Chris من متغير age_dict .

In [104]:
```
# Exercise32 - Answer
del age_dict['Chris']
```

تمرين33. قم بفرز المفتاح بترتيب تصاعدي للمتغير age_dict .

In [105]:
```
# Exercise33 - Answer
sorted(age_dict, reverse=False)
```

Out[105]:
```
['John', 'Lee', 'Lucy']
```

3. عبارة تدفق التحكم

في بايثون ، عبارات تدفق التحكم هي عبارات تستخدم للتحكم في تدفق البرنامج عن طريق تغيير أمر تنفيذه أو عن طريق إعطاء شروط لتنفيذه. توفر بايثون ثلاثة عبارات أساسية للتحكم في التدفق ، وهي عبارات **if** و **while** و **for loops.**
دعنا نتجه لكل منها ونرى كيف تشغيل.

موضوعات

1. **Block Statement**: فهم مفاهيم بيانات الكتلة وبيانات تدفق التحكم
2. **(if Statement(Conditional Statements**: فهم كيفية ومتى استخدام if statement
3. **while Loop**: فهم وظائف while loops
4. **for Loop**: فهم وظائف for loops

[1. عبارة الكتلة]

عبارة الكتلة هي **بيان في كتلة بيثون** ، **مجموعة من نصوص برنامج بايثون** . في بيان تدفق التحكم ، تتكون الكتلة بشكل عام من **وعبارة تدفق التحكم** .
يمكنك أيضًا كتابة كتلة متداخلة عن طريق إدخال كتلة داخل كتلة أخرى.

1.هيكل كتلة بايثون

- يمكن الإشارة إلى كتلة بايثون عن طريق المسافة البادئة.
- هناك طريقتان للمسافات البادئة للسطر ، إحداهما بأربع مسافات بيضاء والأخرى بمسافات بيضاء (مسافة الجدولة). يوصى باستخدام المسافة البادئة ذات الأربع مسافات في بايثون.
[(Reference: PEP 8: Tab or Space) (https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces)](https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces)
- يمكنك رؤية مثال على كتلة بايثون أدناه.

```
# control flow statement
if condition:
    #  block statement indented by 4 whitespaces
    print ("the statement is true.")
```

[2. عبارة شرطية if Statement]

عبارة شرطية هو نوع من عبارات تدفق التحكم المستخدمة لإنشاء كتلة شرطية تعمل فقط عند استيفاء شرط معين.

سننظر في الهيكل الأساسي لعبارات if.

1. الهيكل الأساسي للعبارات الشرطية

- النموذج المعيار للبيان الشرطي هو نموذج `if-elif-else` .
- هنا ، `if` مكون أساسي بينما `elif` و `else` اختياريان.

الهيكل الأساسي على النحو التالي

```
if condition 1: # Inserting the first condition to the program.
    sentence (1) # Sentence (1) and (2) will be executed if the condition
1 is true
    sentence (2)
    ...
elif condition 2: # Inserting the second condition for the case that cond
ition 1 isn't satisfied
    sentence (a) # Sentence (a) and (b) will be executed if the condition
2 is satisfied
    sentence (b)
    ...
else: # If both condition 1 and 2 are false, the below sentence (A) and
 (B) will be executed
    sentence (A)
    sentence (B)
    ...
```

2. نماذج أخرى من عبارات -if

- بالاضافة إلى `if-elif-else` عبارة, العبارات الشرطية يمكن أن تكون `if-else` أو `if` .
- يمكن أن تأخذ العبارات الشرطية أي نوع من أنواع البيانات.

(1) True/False في بيان شرطي

في بيان الشرط المنطقي ، نوع الشرط الأساسي هو

Boolean (True/False).

In [1]:
```python
# If the condition is true,
condition_statement = True

if condition_statement:
    print("The condition is true.")
    print("The conditional statement ends here.")
```

In [2]:
```python
# If the condition is false,
condition_statement = False

if condition_statement:
    print("The condition is true.")
    print("The conditional statement ends here.")
```

إذا `if` بيان كان زائفًا, سيتم تجاهل كتلة if المقابلة دون تنفيذها.

In [3]:
```python
# A comparison operation with Boolean
x = 5
y = 2
if x > y: #  True
    print("x is greater than y.") # Printed
else:
    print("x is less than y") # Not printed
```

(2) True/False في شروط رقمية

بالنسبة إلى البيانات الرقمية ، كل شيء باستثناء 0 سيكون صحيحًا..

- if 0 - False
- if not 0 - True

```
In [4]: if 1:
            print("1 is printed.")

        if 0:
            print("0 is not printed.")

        if -3:
            print("-3 is printed.")
```

(3) True/False في سلاسل

بالنسبة إلى السلاسل ، يكون كل شيء ما عدا السلسلة الفارغة صحيحًا.

- if " " - False
- if "anything" - True

```
In [5]: if "hello~":
            print("True")
```

```
In [6]: if "":
            print("False")
```

[ممارسة] if Statement

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين1. قم بتعيين البيانات المنطقية من صواب إلى متغير `condition` . قم بإخراج السلاسل `This is True` باستخدام if statementو `condition` كشرط.

```
In [7]: # Exercise1 - Answer
        condition = True
        if condition:
            print("This is True!")
```

تمرين2. قم بتعريف متغيرات `num1` و `num2` .تخزين بيانات العدد الصحيح قم بإنشاء عبارة if التي تُخرج السلسلة `They are same` . إسناد هذه العبارة على شرط أن إذا `num1` كان يساوي `num2` , True تم إرجاع

```
In [8]: # Exercise2 - Answer
        num1 = 5
        num2 = 5
        if num1 == num2:
            print("They are same!")
```

تمرين3. قم بخلق العبارة الشرطية التي تتناسب مع شروط أدناه

- قم بتحديد متغير `zero` الذي يخزن 0,ومتغير `empty_str` , الذي يخزن بيانات سلاسل فارغة
- قم باستخدام `zero` كشرط, قم بخلق عبارات ifالتي قامت بإخراج سلاسل `This statement will not be printed!` .
- استخدام متغير `empty_str` كشرط,قم بخلق an elif statement الذي قامت بإخراج سلاسل `This statement will! not be printed`
- أنشئ عبارة else التي تُخرج السلسلة `0 and empty string are False`.

```
In [9]: # Exercise3 - Answer
zero = 0
empty_str = ""

if zero:
    print("This statement will not be printed!")
elif empty_str:
    print("This statement will not be printed!")
else:
    print("0 and empty string are False.")
```

[while Loop .3]

عبارة الحلقة هي نوع من عبارات تدفق التحكم التي تكرر كتلة طالما أن شرطًا معينًا صحيحًا.

عبارة `while` **تكرر** while الكتلة المقابلة **the condition is true**.

1. هيكل أساسي من while Loops

- هنا `شرط` يمكن أن يكون أي شيء يمكنك الحصول على قيمة صواب أو خطأ منه.

الهيكل الأساسي لحلقة while كما يلي.

```
while condition:
    sentence 1
    sentence 2
    sentence 3
    ...
```

لنقم بإنشاء حلقة while للطباعة من 10 إلى 1 واحدًا تلو الآخر.

```
In [10]: counter = 10
while counter > 0:
    print(counter)
    counter = counter - 1
```

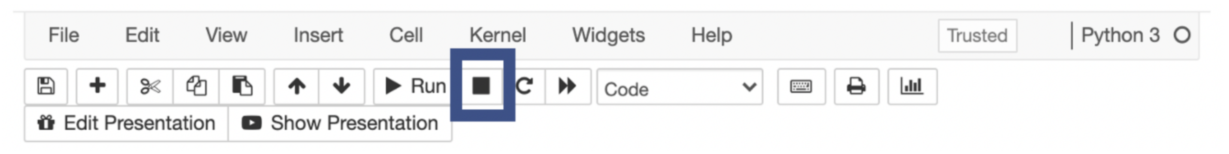
2. نموذج خاص ل while Loop - Infinite Loop

- نظرًا لأن حلقة while تعمل طالما أن الشرط صحيح ، يجب ملاحظة عدم الوقوع في ملف **infinite loop**.

Infinite loop (1)

infinite loopهو عبارة عن كتلة من الحلقات تتكرر إلى ما لا نهاية لأن الشرط لا يزال صحيحًا. في حالة الوقوع في حلقة لا نهائية ، يجب عليك مقاطعة الحلقة بالقلو.

في هذا البرنامج النصي ، يمكنك مقاطعة الحلقة بالزر أدناه.



```
In [11]: # # Running the code below will cause an infinite loop.
         # # The condition is always true and block falls into the infinite loop
         # while True:
         #     print("hello!!!")
```

(2) منع break - infinite loops

لمنع الوقوع في حلقة لا نهائية ، تمتلك بايثون كلمة رئيسية خاصة- break .

نحن نرى كيف break يعمل في infinite loop.

```
In [12]: count = 5

         while True: # `while True` is technically always true and runs endlessly
             print("hello!!!")
             count = count - 1 # 'count' is subtracted by 1 at every iteration
             if count == 0: # once the 'count' reaches 0, it escapes the while loop using
                 break
```

(3) إنهاء التكرار الحالي - continue

continue هو كلمة أساسية أخرى في بايثون تُستخدم لإنهاء التكرار الحالي وإعادة عنصر التحكم إلى بداية حلقة while

```
In [13]: count = 5

         while count:
             count = count - 1 # 'count' subtracted by one at every execution

             if count % 2 == 0:
                 # 'count' is false at every odd number and true at every even number
                 # The execution of `print` statement will be ignored at every even-number
                 continue
             print("counter : ", count)
```

[ممارسة] while Loop

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين4. قم بانشاء a while loop الذي يناسب مع شروط أدناه

- قم بتصريح متغير count1 الذي يخزن القيمة0that stores the value 0.
- قم بانشاء create a while loop الذي الحلقة إذا قيمة count1 كان تحت 5
- قم بإخراج قيمة من count1 لكل loop.
- لكل loop, قيمة count1 يتم زيادته بمقدار 1.

```
In [14]: # Exercise4 - Answer
         count1 = 0
         while count1 < 5:
             print(count1)
             count1 = count1 + 1
```

تمرين5 قم بانشاء while loop الذي يناسب مع شروط

- قم بانشاء متغير count2 الذي يخزن 1.
- اضبط الشرط لـwhile loop كصحيح
- لكل loop, قيمة count2 يتم زيادته بمقدار 1
- عندما تكون قيمة count2 رقمًا فرديًا ، فانتقل إلى الحلقة التالية باستخدام الكلمة الرئيسية للمتابعة. إذا كانت القيمة عددًا زوجيًا ، فقم بإخراج قيمة count2 .
- عندما تكون قيمة count2 تساوي 10 ، قم بإنهاء الحلقة باستخدام الكلمة الأساسية break.

In [15]:
```python
# Exercise5 - Answer
count2 = 1
while True:
    count2 = count2 + 1

    if count2 % 2 == 0:
        print(count2)
    else:
        continue

    if count2 == 10:
        break
```

[4. for Loop]

for loop نوع من عبارات تدفق التحكم التي **تتكرر عبر تسلسل أو متكرر (قائمة ، أو مجموعة ، أو قاموس ، أو مجموعة أو سلسلة)** .
للاحظ أنه يتم استخدام المصطلحين التسلسل و التكرار كمرادفات في هذا الدرس.

1.هيكل أساسي for Loops

الهيكل الأساسي للحلقةfor هو كما يلي.

```python
for element in iterable(e.g. list, tuple, tuple, string, dictionary, etc
):
    sentence 1
    sentence 2
    ...
```

في السطر الأول من الحلقة التي تحتوي على تعليمة for ، يمكن أن يكون التكرار عبارة عن قائمة ، أو مجموعة ، أو مجموعة ، أو مجموعة.
تكرر حلقة `for` على عناصر قابلة للتكرار واحدة تلو الأخرى.

2. مثال for Loop

- سنلقي نظرة على مثال حلقة for أساسية.

دعونا نصنع حلقة for للفواكه في محل بقالة

In [16]:
```python
products = ["banana", "apple", "mango", "pineapple"]

for product in products:
    print(product)
```

(1) هروب للحلقة- break

باستخدام break , يمكنك إنهاء حلقة for عند نقطة معينة بدلاً من تشغيلها حتى النهاية. سنقوم بطباعة أسماء الفاكهة حتى تتم طباعة اسم mango .

In [17]:
```python
products = ["banana", "apple", "mango", "pineapple"]

for product in products:
    print(product)
    if product == "mango":
        break
```

```
banana
apple
mango
```

توقفت الحلقة بعد طباعة mango .

(2) إنهاء التكرار الحالي - continue

ينهي continue التكرار الحالي ويعيد عنصر التحكم إلى بداية حلقة for.
دعونا نرى كيف يمكنك طباعة أسماء الفاكهة باستثناءاستخدام التفاح continue .

In [18]:
```
products = ["banana", "apple", "mango", "pineapple"]

for product in products:
    if product == "apple":
        continue
    print(product)
```

```
banana
mango
pineapple
```

3. الهياكل الأخرى for Loops

(1)التكرار على القاموس

يتكون قاموس بايثون من أزواج مفتاح - قيمة. يمكنك تكرار المفاتيح والقيم بشكل منفصل أو متزامن ، كيفما تريد.

In [19]:
```
fruits_price = {
    "banana" : 20,
    "apple" : 10,
    "mango" : 25,
    "pineapple" : 35
}
```

بشكل افتراضي ، تتكرر حلقات for بناءً على قيم المفاتيح.

In [20]:
```
for price in fruits_price:
    (price)
```

للتكرار على القيم ، أضف values. بعد اسم القاموس.

In [21]:
```
for price in fruits_price.values():
    print(price)
```

```
20
10
25
35
```

أو ، للتكرار على المفاتيح والقيم ، أضف items بعد اسم القاموس

In [22]:
```
print(fruits_price.items())
```

```
dict_items([('banana', 20), ('apple', 10), ('mango', 25), ('pineapple', 35)])
```

(2) التكرار على تعديدات

تتشابه تعديدات بشكل أساسي مع قوائم بايثون ، ولكن على عكس القوائم ، فإن تعديد غير قابلة لتغيير.

```
In [23]: products = ("banana", "apple", "mango", "pineapple")

         for product in products:
             print(product)
```

```
banana
apple
mango
pineapple
```

for Loop [ممارسة]

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين6. قم بتعريف قائمة متغير colors الذي يخزن red, yellow, green, blue . باستخدام حلقة for ، أخرج جميع قيم السلسلة المخزنة في color .

```
In [24]: # Exercise6 - Answer
         colors = ["red", "yellow", "green", "blue"]

         for color in colors:
             print(color)
```

```
red
yellow
green
blue
```

تمرين7. استخدام بيانات سلاسل red, yellow, green, blue كمفتاح,و apple, banana, cucumber, blueberry كقيم للمفاتيح ، حدد متغير القاموس color_fruits. باستخدام حلقة for ، قم بإخراج جميع القيم باستثناء cucumber .

```
In [25]: # Exercise7 - Answer
         color_fruits = {"red":"apple", "yellow":"banana", "green":"cucumber", "blue":"blu
         for fruit in color_fruits.values():
             if fruit == "cucumber":
                 continue

             print(fruit)
```

```
apple
banana
blueberry
```

تمرين8. قم بتعريفtuple متغير find_cucumber الذي يخزن apple, banana, cucumber, blueberry . باستخدام حلقة for ، أخرج كل السلسلة حتى cucumber في , find_cucumber ثم قم بإنهاء الحلقة.

In [26]:

```
# Exercise8 - Answer
find_cucumber = ("apple", "banana", "cucumber", "blueberry")

for fruit in find_cucumber:
    print(fruit)

    if fruit == "cucumber":
        break
```

```
apple
banana
cucumber
```

4. وظائف وحزم بايثون

في البرمجة ، هناك أوقات تظهر فيها نفس سطور التعليمات البرمجية بشكل متكرر بحيث تجدها مملة ومرهقة. لتجنب مثل هذه اللحظات غير الفعالة ، توفر بايثون مجموعة من الأدوات الرائعة. إنها `وظائف` لإعادة استخدام التعليمات البرمجية ، و `حزم` وهي مجموعات من الوظائف. بمجرد أن تتعلم متى وكيف تستخدم الوظائف والحزم ، ستتمكن من إدارة عبء العمل بشكل أكثر ذكاءً من عدم استخدامها. لذلك ، في هذا الدرس ، سنركز على فهم المفاهيم واستخدام الوظائف والحزم.

موضوعات,

1. **وظائف بايثون**: فهم ما هي الوظائف وكيفية استخدامها
2. **حزم بايثون**: فهم ما هي الحزم وكيفية استخدامها

[1. وظائف بايثون]

في بايثون ، الوظيفة عبارة عن مجموعة من العبارات ذات الصلة التي تؤدي مهمة محددة. تشبه وظيفة بايثون الوظائف في الرياضيات في العديد من الجوانب لأنها تأخذ مُدخل `x` ، وتعالجها من خلال دالة `f` وتُرجع ناتجًا `f(x)` تمامًا كما هو الحال في الرياضيات. على الرغم من ذلك ، فإن الإدخال والإخراج لوظيفة بايثون اختيارية بينما لا يكون الأمر كذلك في الرياضيات.

1. إنشاء الوظيفة

- غالبًا ما يُشار إلى إنشاء وظيفة باسم "تحديد وظيفة.
- في بايثون ، تُستخدم الكلمة الأساسية `def` لتمييز بداية الوظيفة ويتبعها اسم الوظيفة كاسم الكتلة.
- اقم بتحديد معلمات الإدخال وقيم الإرجاع في وظيفة 2 (اختياري).

(1) تعريف الوظائف

قبل أن ننتقل إلى تعريف الوظيفة ، دعنا نلقي نظرة على ماهية الوظيفة وكيف تبدو.

- اسم الوظيفة (**مطلوب**) - `add`
- بارامتر (ادخال)(**اختياري**) - `a, b`
- إرجاع العبارة(إخراج) (**اختياري**) - `return`

In [1]:
```python
def add(a, b):
    result = a + b
    return result
```

في الوظيفة أعلاه ، تمت تسمية الوظيفة `add` بعد الكلمة الأساسية `def`. يعمل اسم الوظيفة كمعرف للاستخدام لاحقًا. بعد ذلك ، يتم تحديد بارمترتان `a` و `b` ، يمكن للوظيفة أن تأخذ قيمتي إدخال بواسطتها. في السطر الثاني ، تتم إضافة قيمتي الإدخال `a` و `b` وتخزينهما في المتغير `result`. في السطر الأخير ، تشير الكلمة الأساسية `return` إلى أن الوظيفة ستعيد قيمة مخرجات `result` ، في هذه الحالة ، عند تنفيذها.

(2)تسمية الوظيفة

دعنا نسمي الوظيفة التي أنشأناها للتو.

In [2]:
```python
y = add(3,5) # Passing two integers, 3 and 5, to 'a' and 'b' respectively
```

وظيفة `add` تأخذ 3 و 5 كقيم إدخال وإرجاع 8 كقيمة الإخراج.

تسمى القيم المدرجة في المعلمات ، 3 و 5 ، **الوسيطات** يقوم استدعاء وظيفة بإرجاع خطأ عندما لا يتم إدراج وسيطات إلى وظيفة ذات معلمات

In [3]:
```python
# This code will cause an error.
# add()
```

اذا قمت بتنفيذ التعليمات البرمجية أعلاه,فستحصل على رسالة خطأ كما هو أدناه

مثال رسالة الخطأ)

NameError Traceback (most recent call last) in ----> 1 add()

NameError: name 'add' is not defined ```

(3) الوظائف قاعدة على المعلمات وقيم الإرجاع

تأخذ الوظيفة `add` وسيطات وتُرجع قيمة الإخراج. إذا كنت تنوي عدم تحديد أي منها ، فما عليك سوى ترك المعلمات وعبارة `return` خارج وظيفة.

وظيفة بدون معلمات وقيمة إرجاع

In [4]:
```
def make_a_sound(): # Defining a function without a parameter
    print("Meow")  # Function without a return value but a print statement.


y = make_a_sound()# Calling the function
```

```
Meow
```

على الرغم من أن `make_a_sound ()` تطبع `Meow` ، فإن القيمة المطبوعة ليست هي القيمة التي ترجعها الدالة لأن الوظيفة لا تحتوي على بيان إرجاع. لذلك ، لن تحتفظ `y` بأي قيمة حتى بعد التنفيذ.

In [5]:
```
y # Returns nothing(`None`) indicating there is no returned value stored in `y`
```

وظيفة بايثون بدون معلمات

In [6]:
```
def agree(): # Defining a function named `agree` with no parameter
    return True # But a return value

y = agree() # Calling the `agree` function
```

In [7]:
```
y
```

Out[7]:
```
True
```

وظيفة بايثون بدون عبارة الإرجاع

In [8]:
```
def hello_by(name): # Defining a function named `hello_by` with a parameter
    print("Hello, {}".format(name)) # Print statement without a return value

y = hello_by("John") # Calling the function
```

```
Hello, John
```

[ممارسة] إنشاء الوظيفة

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين1. باستخدام متغيرين `num1` و `num2` كمعلمات ، حدد وظيفة تسمى `sub` تعرض `num 1` مطروحًا من `num 2`

In [9]:
```
# Exercise1 - Answer
def sub(num1, num2):
    print(num1 - num2)
```

تمرين2. عيّن 2 و 3 إلى `test1` و `test2` على التوالي ، ثم استدع الوظيفة `sub` وتحقق من النتيجة.

In [10]:
```python
# Exercise2 - Answer
test1 = 3
test2 = 2
sub(test1, test2)
```

```
1
```

تمرين3. حدد وظيفة **python_function** التي تُخرج السلسلة النصية **This is a python function** وتعيد السلسلة **This is a return value** .

In [11]:
```python
# Exercise3 - Answer
def python_function():
    print("This is a python function")
    return  "This is a return value"
```

تمرين4. قم باستدعاء الوظيفة **python_function** ، وقم بتعيين النتيجة إلى المتغير **return_value** .

In [12]:
```python
# Exercise4 - Answer
return_value = python_function()
return_value
```

```
This is a python function
```

Out[12]:
```
'This is a return value'
```

أنواع وظائف بايثون

- بالإضافة إلى الوظائف التي يحددها المستخدم مثل ما قمت بإنشائه مسبقًا في هذا الدرس ، تدعم بايثون الوظائف المضمنة بالإضافة إلى الوظائف المجهولة

(1) الوظائف المضمنة

توفر بايثون عددًا من الوظائف المضمنة . يتم تثبيت الوظائف المدمجة تلقائيًا على جهاز الكمبيوتر الخاص بك في وضع جاهز للاستخدام عند تثبيت بايثون.

sum - يضيف قيم الإدخال

In [13]:
```python
lst1 = [1,2,3,4,5]
sum(lst1) #
```

Out[13]:
```
15
```

max -تُرجع القيمة القصوى لقيم الإدخال

In [14]:
```python
lst1 = [1,2,3,4,5]
max(lst1)
```

Out[14]:
```
5
```

min - تُرجع أصغر قيمة لقيم الإدخال

In [15]:
```
lst1 = [1,2,3,4,5]
min(lst1)
```
Out[15]: 1

abs - تُرجع القيمة المطلقة لقيمة الإدخال.

In [16]:
```
abs(-1.7)
```
Out[16]: 1.7

map - تطبق الوظيفة المعينة على كل عنصر من عنصر تكراري معين.

In [17]:
```
def add_1(x): # Defining a function named `add_1`
    return x + 1

lst1 = [1, 2, 3, 4, 5]
list(map(add_1, lst1)) # Applying `add_1` to each element of `lst1`
```
Out[17]: [2, 3, 4, 5, 6]

filter -يختبر ما إذا كان كل عنصر من عناصر التكرار يفي بشروط معينة ويعيد العناصر التي تلبيها فقط

In [18]:
```
def is_even(x):
    return x % 2 == 0

lst1 = [1,2,3,4,5]
list(filter(is_even, lst1)) # Tests if elements of `lst1` meet `is_even` and retu
```
Out[18]: [2, 4]

(2) الوظيفة المجهولة - `lambda`

إلى جانب الوظائف المضمنة والمعرفة من قبل المستخدم ، تدعم بايثون الوظائف غير المعنونة والتي تسمى الوظائف المجهولة . الوظيفة المجهولة هي وظيفة لمرة واحدة ويتم تعريفها بإيجاز شديد في سطر واحد مع الكلمة الأساسية `lambda` في البداية. نظرًا لأنه يمكن أن يوفر الكثير من وقت عملك ، فمن الأهمية بمكان معرفة وقت وكيفية استخدامه. إذن، دعونا نحاول إنشاء واحدة.

In [19]:
```
def add(x, y):
    return x + y

add(1, 2)
```
Out[19]: 3

يمكن كتابة الوظيفة أعلاه على النحو التالي.

In [20]:
```
"""
the function itself doesn't have a name
and should be stored to a variable for later reuse.
"""
add_by_lambda = lambda x, y : x + y
add_by_lambda(1, 2)
```
Out[20]: 3

In [21]:
```
add_by_lambda(2, 3)
```
Out[21]: 5

يتم استخدام lambda للإشارة إلى نقطة البداية للوظيفة المجهولة. يتبعها نقطتان ومحتوى الوظيفة.

In [22]:
```
def add_1(x):
    return x + 1

lst1 = [1,2,3,4,5]
list(map(add_1, lst1))
```

Out[22]:
```
[2, 3, 4, 5, 6]
```

In [23]:
```
lst1 = [1,2,3,4,5]
list(map(lambda x : x + 1, lst1))
```

Out[23]:
```
[2, 3, 4, 5, 6]
```

[ممارسة] أنواع وظائف بايثون

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين5. اكتب الكود لكل التعليمات أدناه.

- قم بتصريح متغير قائمة `lst2` الذي يخزن بيانات عدد صحيح [1,2,3,4,5,6].
- باستخدام الوظائف المضمنة ، استرجع القيمة القصوى في عناصر القائمة ، ثم خزن القيمة في متغير `lst_max` .
- باستخدام الوظائف المضمنة ، استرجع القيمة الدنيا في عناصر القائمة ، ثم خزن القيمة في متغير `lst_min` .
- اعرض مجموع `lst_max` و `lst_min` .

In [24]:
```
# Exercise5 - Answer
lst2 = [1,2,3,4,5,6]
lst_max = max(lst2)
lst_min = min(lst2)
print(lst_max + lst_min)
```

```
7
```

تمرين6.أكتب الكود لكل التعليمات أدناه.

- قم بتعريف متغير `ab1` الذي يخزن بيانات عدد صحيح -10 ومتغير `ab2` الذي يخزن 6.
- باستخدام الوظيفة المضمنة ، حدد وظيفة lambda المجهولة `abs_add` التي تحسب القيم المطلقة للمعاملين x و y ، ثم تعرض مجموع القيم المطلقة.
- تمرير `ab1` و `ab2` كوسيطات لوظيفة `abs_add` lambda ، إخراج النتيجة.

In [25]:
```
# Exercise6 - Answer
ab1 = -10
ab2 = 6
abs_add = lambda x, y : abs(x) + abs(y)
print(abs_add(ab1, ab2))
```

```
16
```

تمرين7.أكتب الكود لكل التعليمات أدناه.

- حدد متغيرقائمة `prime_numbers` الذي يخزن بيانات عدد صحيح [2,3,5,7,11].
- قم بتعريف متغير big_than_5 يأخذ المعلمة x ويعيد True إذا كانت قيمة x أكبر من 5.
- باستخدام الوظيفة المضمنة ، قم بإرجاع عناصر `prime _numbers` التي تتناسب مع شرط الوظيفة `bigger _than_5` كقائمة. إخراج القائمة.

```
In [26]: # Exercise7 - Answer
prime_numbers = [2,3,5,7,11]
def bigger_than_5(x):
    return x > 5
print(list(filter(bigger_than_5, prime_numbers)))

[7, 11]
```

[2. حزم بايثون]

الحزمة عبارة عن مجموعة من الوحدات والوظائف والأساليب. تم تطويره من قبل مطورين آخرين على أمل مساعدة مستخدمي بايثون الآخرين في كتابة التعليمات البرمجية الخاصة بهم بطريقة أكثر كفاءة وفعالية. تدار في هيكل دليل. تشبه الحزم مجموعات موضوعية من الوحدات وتجد كل حزمة استخدامًا في مجال مختلف من المهام.

على سبيل المثال ، سيتم تقديمك إلى ثلاث حزم مختلفة لاحقًا في هذا البرنامج ، وهي نمباي و باندا و مات بلوت ليب للحساب الرياضي للبيانات ومعالجة البيانات وتصور البيانات على التوالي. قبل أن ننظر إلى أبعد من ذلك ، سنلقي نظرة على بعض حزم بايثون الأساسية.

1. استخدام حزم بايثون

بشكل افتراضي ، يؤدي استيراد حزمة بكلمة رئيسية `import` إلى تحميل محتويات الحزمة بالكامل إلى ملف العمل الحالي. من أجل استخدام وحدة نمطية ، يجب أن تسبق اسم الحزمة قبل اسم الوحدة النمطية `[package].[module]` .

- ومع ذلك ، إذا كنت تنوي import وحدات أو طرق معينة فقط ، فيجب عليك استيرادها بواسطة `from import` النحو مثل `from [package] import [modules]` .
- في حال كان اسم الحزمة طويلًا جدًا أو تم الوصول إليه كثيرًا ، يمكنك تسميته.

(1)استيراد الحزمة أكملها

قيل أنه يمكنك استيراد حزمة باستخدام الكلمة الأساسية `import` . بدلاً من ذلك ، يمكنك التعبير صراحةً عن أنك تقوم باستيراد كل شيء من الحزمة باستخدام `from package_name import *` .

علامة النجمة(*) هي حرف بدل يشير إلى `all` . باستخدام بناء الجملة هذا ، يمكنك استخدام الوظائف الكاملة للحزمة المستوردة مباشرةً دون إضافة بادئة إلى اسم الحزمة.

```
In [27]: from random import * # Importing the entire contents of the `random` package
```

```
In [28]: lst1 = [1,2,3,4,5]
print(choice(lst1)) # Using `choice()` of `random` without prefixing the package

4
```

```
In [29]: shuffle(lst1) # Using `shuffle()` from the `random` without the package name
print(lst1)

[2, 5, 3, 1, 4]
```

```
In [30]: rand_int = randint(1,100) # Using `.randint()` from the `random` without the pack
print(rand_int)

83
```

(2)استيراد وحدات معينة من الحزمة

ستكون هناك أوقات تريد فيها استيراد جزء معين فقط من الحزمة في ملفك. على سبيل المثال ، إذا كنت تحتاج فقط إلى الوظيفة `shuffle` من الحزمة `random` ، فاستوردها كـ `from random import shuffle` . تقول صراحةً أنك تستورد `shuffle` فقط من الحزمة `random` .

```
In [31]: from random import shuffle
```

In [32]:
```
lst1 = [1,2,3,4,5]
shuffle(lst1)
lst1
```
Out[32]:
```
[3, 1, 4, 2, 5]
```

أو ، إذا كنت بحاجة إلى استيراد وحدتين أو أكثر من الحزمة ، فقم بما يلي.

In [33]:
```
from random import choice, randint # Importing choice and randint from the `rando
```

In [34]:
```
lst1 = [1,2,3,4,5]
choice(lst1)
```
Out[34]:
```
1
```

(3) استيراد حزمة باسم مستعار

إذا تم استدعاء اسم الحزمة كثيرًا أو طويلًا جدًا بحيث لا يمكن كتابته يدويًا بشكل متكرر ، فهناك طريقة لاستدعائه بشكل أكثر ملاءمة ، أي عن طريق تسمية اسم الحزمة بشكل مستعار. في اصطلاح بايثون ، يُطلق على `numpy` اسم `np` أو `pandas` كـ `pd` أو `BeautifulSoup` كـ `bs` ، ويتم تسمية هذه الحزم الخارجية باسم مستعار أثناء استيرادها كما هو موضح أدناه.

In [35]:
```
# Importing 'numpy' under 'np'
import numpy as np
# np.add()
lst1 = [1,2,3,4,5]
np.add(lst1, 1)
```
Out[35]:
```
array([2, 3, 4, 5, 6])
```

[ممارسة] حزم بايثون

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

تمرين8. من حزمة `random` ، قم باستيراد طريقة `gauss` فقط.

In [36]:
```
# Exercise8 - Answer
from random import gauss
```

تمرين9.قم باستيراد حزمة `matplotlib.pyplot` كاسم مستعار `plt`.

In [37]:
```
# Exercise9 - Answer
import matplotlib.pyplot as plt
```

2.حزمة بايثون القياسية

- توفر بايثون عددًا من الحزم القياسية أيضًا.
- أمثلة على حزمة بايثون القياسية هي `time`. دعونا نلقي نظرة على كيفية استخدامها.

(1) `time`

`time` هو حزمة تستخدم لقياس الوقت. وظيفة `time()` من حزمة `time` تعرض وقت نظام التشغيل الحالي بالثواني. لنلقي نظرة على مثال.

In [38]:
```
import time # Imoprting `time` package
```

In [39]:
```
time.time() # Returns the current OS time in seconds
```

Out[39]:
```
1621211792.7155871
```

os (2)

os عبارة عن حزمة تسمح باستخدام العديد من الوظائف التي يوفرها os (نظام التشغيل).

In [40]:
```
import os
```

باستخدام `listdir` من نظام التشغيل ، يمكنك الحصول على أسماء جميع الملفات في الدليل المحدد.

In [41]:
```
# Reading all file names in the current directory
os.listdir("./")
```

Out[41]:
```
['2. Data Structure.ipynb',
 '3. Control Flow Statement.ipynb',
 '4. Python Functions and Packages.ipynb',
 '1. Variables and operations .ipynb',
 '.ipynb_checkpoints',
 '5. Object-Oriented Programming (OOP)-Python Classes and Class Inheritance.ipy
nb']
```

In [42]:
```
# Reading all file names in the parent directory
os.listdir("../")
```

Out[42]:
```
['3. translated-final',
 '1. translated',
 '.DS_Store',
 '2. edited',
 '4. edited-form',
 '.ipynb_checkpoints']
```

5.البرمجة الموجهة للموجودات - الفئات والوراثة في بايثون

الفئة عبارة عن قالب رمز لإنشاء الموجودات. في بايثون، يتم تعريف الفئة بواسطة الكلمة الأساسية `فـئـة` في البداية ، ويتكون الموجود من متغيرات الأعضاء والسلوكيات المرتبطة بهذه المتغيرات. قد يبدو هذا غامضًا. لذلك ، في هذا القسم ، سوف نلقي نظرة على تعريف وخصائص فئات بايثون والأشياء ثم الوراثة والفئات.

موضوعات

1. **فئات وأمثلة**: فهم تعريف وخصائص الفئات والمثيلات وأدوات OOP الأخرى في بايثون
2. **وارثة الفئات**: فهم كيفية استخدام وراثة الفئات لكتابة رمز بطريقة أكثر كفاءة

[1. فئات وأمثلة]

الفئة عبارة عن قالب رمز لإنشاء موجودات وتتكون من `سـمـات و طـرق` . تعمل السمة في الفئة بشكل مشابه لمتغير بايثون كحامل للقيمة ، وتعمل الطريقة كدالة في بايثون لأنها تؤدي مهمة معينة.

بمجرد تحديد فئة ، يمكنك إنشاء موجودات بناءً على وظائف الفئة المحددة ، ويسمى هذا الموجود الفردي لفئة معينة مثيلاً.

ضع في اعتبارك أن السمات تشبه المتغيرات والطرق تشبه الوظائف.

1. هيكل فئات بايثون

لنقم بإنشاء فئة بايثون لرؤية الهيكل

(1) تعريف الفئة

يبدأ تعريف الفئة بالكلمة الأساسية `فـئـة` . ثم يتبعه اسم الفئة ونقطتان (`:`). تنشئ النقطتان كتلة للفئة من السطر التالي وهناك يمكنك تحديد وظيفة الفئة لاحظ أن اسم الفئة يبدأ بحرف كبير

للمساعدة في فهم أفضل ، سننشئ فئة باسم `Wizard` .

In [1]:
```python
class Wizard:
    pass
```

تم إنشاء `Wizard` الآن

In [2]:
```python
Wizard
```

Out[2]:
```
__main__.Wizard
```

(2)سمات الفئة

سمات الفئة هي متغيرات بايثون التي تنتمي إلى فئة. يتم تعريفها خارج مُنشئ الفئة (والتي ستتعلمها لاحقًا في هذا الدرس) وتتم مشاركتها بواسطة جميع موجودات هذا الفصل.

لنفترض هنا أن هناك ساحرًا يمارس السحر بعصا سحرية.إذن هيا نضيف `Magic wand` إلى سمات الفئة `weapon` داخل فئة `Wizard` .

In [3]:
```python
class Wizard:
    weapon = "Magic wand"
```

عند الاستدعاءأو الوصول إلى سمة فئة ، يجب أن تسبق اسم الفئة للإشارة إلى أنك تصل إلى تلك الفئة المعينة. يشير وجود نقطة بين اسم الفئة واسم سمة الفئة إلى أنك تصل إلى تلك السمة المعينة للفئة.

هيا نقوم باستدعاء `weapon` من فئة `Wizard`

```
In [4]: Wizard.weapon # Every instance of the `Wizard` class has a `Magic wand` as a weap
```

```
Out[4]: 'Magic wand'
```

(3) سمات المثيل

كما ذكر أعلاه ، المثيل هو موجود تم إنشاؤه بناءً على فئة معينة. تشير سمة المثيل إلى المتغيرات الفريدة للمثيل التي لا تتم مشاركتها مع مثيلات أخرى ، بخلاف سمات الفئة.

يمكن تعريف سمات المثيل **inside the constructor method** , `__ init __ (self,..)` ، من فئة معينة ويتم الوصول إليها فقط في نطاق مثيلاتها. نظرًا لعدم مشاركة قيم سمات المثيل بين المثيلات ، فإن كل مثيل يخزن قيمه الفريدة لسمات المثيل نفسها.

Constructor

`constructor` هو نوع خاص من الطرق التي يتم استدعاؤها تلقائيًا عند إنشاء كل مثيل.
كل مُنشئ يأخذ `self` كأول معلمة للإشارة ضمنيًا إلى المثيل نفسه. عند استدعاء مثيل ، لا تحتاج إلى تمرير وسيط إلى المعلمة `self` .

```
In [5]: class Wizard:
            weapon = "Magic wand"

            def __init__(self, name, sex, year): # Constructor
                self.name = name # Instance attribute
                self.sex = sex # Instance attribute
                self.year = year # Instance attribute
```

الآن ، يمكننا تخزين الاسم والجنس وسنة الميلاد لكل عنصر يشترك في فئة `Wizard`

```
In [6]: # `self` takes the instance `is my_wizard `.
        # The input arguments are passed to `name`, `sex`, and `year` respectively
        my_wizard = Wizard('magician1', 'male', '2020')
```

الساحر:wizard*
سمات مثيل `my_wizard` هي كما يلي.

```
In [7]: print(my_wizard.name)
        print(my_wizard.sex)
        print(my_wizard.year)

magician1
male
2020
```

يحتوي my_wizard2 على معلومات حول الساحر ، وهو شخصية ذكورية تم إنشاؤها في عام 2020.

```
In [8]: my_wizard2 = Wizard('magician2', 'female', '2020')
```

```
In [9]: print(my_wizard2.name)
        print(my_wizard2.sex)
        print(my_wizard2.year)

magician2
female
2020
```

يحتوي my_wizard2 في الساحر 2 على معلومات. الساحر 2 هي شخصية أنثوية تم إنشاؤها في عام 2020.
يستخدم كل من الساحر 1 و 2 عصا سحرية كسلاح ، وهنا ، العصا السحرية هي سمة من سمات فئة `wizard`

```
In [10]: my_wizard.weapon
```

```
Out[10]: 'Magic wand'
```

In [11]:
```
my_wizard2.weapon
```
Out[11]:
```
'Magic wand'
```

(4) أساليب الفئة

الأسلوب هو دالة معرفة داخل كتلة فئة ولا يمكن استخدامها إلا للحالات التي تشترك في فئة معينة.
الساحر فئة للأشياء بالسحر ، ويجب أن تتمتع موجوداتها بمهارات التعافي من أجل الصحة والمانا.
دعونا ننشئ طريقتين للفصل لكل منهما في `Wizard`

In [12]:
```
class Wizard:
    weapon = "Magic wand"
    def __init__(self, name, sex, date):
        self.name = name
        self.sex = sex
        self.date = date
    """
    `@staticmethod` to denote a class method
    that doesn't take a compulsory parameter
    """
    @staticmethod
    def health():
        print(f"{Wizard.weapon} can recover health")
    """
    `@classmethod` to denote a class method that takes compulsory parameters
    with `cls` as the first parameter.
    """
    @classmethod
    def mana(cls):
        print(f"{cls.weapon} can recover mana")
```

In [13]:
```
Wizard.health()
```
```
Magic wand can recover health
```

In [14]:
```
Wizard.mana()
```
```
Magic wand can recover mana
```

كما ترى أعلاه ، هناك نوعان من الطرق في الفصل ، طريقة ثابتة وطريقة فئة. الفرق هو أن `@ classmethod` الوصول إلى حالة الفئة أو تعديلها بينما لا يمكن للطريقة الثابتة.

(5) أساليب المثيل

أساليب المثيل هي الأساليب التي لا يمكن استخدامها إلا في حالات الفئة حيث يتم تعريف الطرق. تعتمد خاصية المثال على نوع المهارات التي يتعلمها كل ساحر. لنمنح الساحر 1 مهارة تسمى `fireball`

In [15]:
```
class Wizard:
    weapon = "Magic wand"
    def __init__(self, name, sex, date):
        self.name = name
        self.sex = sex
        self.date = date
    @staticmethod
    def health():
        print(f"{Wizard.weapon} can recover health")
    @classmethod
    def mana(cls):
        print(f"{cls.weapon} can recover mana")

    def fireball(self):
        print(f"{self.name} learned skill 'fire ball'")
```

In [16]:
```python
my_wizard = Wizard('magician1', 'male', '2020')
```

In [17]:
```python
my_wizard.fireball()
```

```
magician1 learned skill 'fire ball'
```

[ممارسة] فئات وأمثلة

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.

قم بإنشاء فئة لإنشاء مثيل لشيء يحتوي على معلومات عن شركة.

تمرين1. أنشئ فئة باسم `Company` وطريقة منشئ تخزن `id` و `name` و `owner` .

In [18]:
```python
# Exercise1 - Answer
class Company():
    def __init__(self, id, name, owner):
        self.id = id
        self.name = name
        self.owner = owner
```

تمرين2. قم بإنشاء طريقة تسمى `print_info` تقوم بإخراج معلومات الشركة(`name` , `id` , و `owner`) في فئة `Company` .

ID : `id`
Name : `name`
Owner : `owner`

In [19]:
```python
# Exercise2 - Answer
class Company():
    def __init__(self, id, name, owner):
        self.id = id
        self.name = name
        self.owner = owner

    def print_info(self):
        print(f"ID : {self.id}")
        print(f"Name : {self.name}")
        print(f"Owner : {self.owner}")
```

تمرين3. إنشاء مثيل لعنصر شركة لفئة `company` باستخدام المعلومات الواردة أدناه ، وتعيينه إلى المتغير `company1` .

- **`id` : 1234**
- **`name` : company1**
- **`owner` : John**

In [20]:
```python
# Exercise3 - Answer
company1 = Company(1234, 'company1', 'John')
```

تمرين4. قم بإخراج `name` , `id` و, `owner` من `company1` .

In [21]:
```
# Exercise4 - Answer
print(company1.id)
print(company1.name)
print(company1.owner)
```

```
1234
company1
John
```

تمرين5. قم بتنفيذ **print_info** أسلوب في **company1** .

In [22]:
```
# Exercise5 - Answer
company1.print_info()
```

```
ID : 1234
Name : company1
Owner : John
```

[2.وراثة الفئة]

لتجنب كتابة نفس سطور التعليمات البرمجية بين الفئات مرارًا وتكرارًا ، يمكنك استخدام ميزة من فئة بايثون تسمى `وراثة الفئة` . كما يشير الاسم ، إنها ميزة تمكّن فئة ما من أن ترث من فئة أخرى لقابلية إعادة استخدام الكود وقابليته للقراءة. يُطلق على الفئة الموروث منه فئة الأصل والفئة التي ترث من فئة أصل هي فئة فرعية. تشترك فئة الأصل في خصائصها ووظائفها مع الفئة الفرعي.

1. وراثة الفئة من فئة الوالدين

- تستخدم فئات بايثون وراثة الفئة لزيادة إمكانية إعادة استخدام الكود.
- من بين العديد من فئات بايثون ، تعتبر فئة `object` هي الفئة الأساسية في بايثون وتوفر الكثير من الوظائف ، وترث جميع الفئات في بايثون 3 من `object` .

مثال) إنشاء فئة من خلال شخصية اللعبة

نقوم بإنشاء فصول لكل من الفارس والساحر والرامي ، وهي تحتوي بالضبط على نفس الأساليب.

In [23]:
```
# If not using class inheritance,
class Knight:
    def move(self):
        print("You can move.")

    def attack(self, food):
        print("Attack the enemy.")

class Wizard:
    def move(self):
        print("You can move.")

    def attack(self, food):
        print("Attack the enemy.")

class Archer:
    def move(self):
        print("You can move.")

    def attack(self, food):
        print("Attack the enemy.")
```

من أجل منع تكرار التعليمات البرمجية ، سنقوم بتعريف الطرق المتكررة في فئة جديدة تسمى `person` ونجعل `Knight` و `Archer` و `wizard` يرثون من `person` .

In [24]:
```
# Defining `the parent class `Person`
class Person():
    def move(self):
        print("You can move.")
```

بمجرد تحديد `person` ، سننشئ فئات فرعية ترث من `person` .

In [25]:
```
# Class inheritance from `person`
class Knight(Person):
    def sword(self):
        print("The sword is available.")

class Wizard(Person):
    def magic_wand(self):
        print("The magic wand is available.")

class Archer(Person):
    def bow(self):
        print("The bow is available.")
```

2. أساليب الوراثة من فئة الوالدين

- يرث الصنف الفرعي المجموعة الكاملة من الطرق من الصنف الأصل.
- في حالة تحديد طريقة في فئة فرعية تحت نفس الاسم لاسم طريقة في الفئة الأصلية ، فإن الطريقة ستؤدي المهمة المحددة في الفئة الفرعية ، وتسمى overriding. باختصار ، يعني التجاوز استبدال تنفيذ طريقة محددة في فئة رئيسية بالطريقة الجديدة في الفئة الفرعية.

(1) الوراثة

ستتلقى الفئة الفرعية التي ترث من فئة أصل الطرق المحددة في الفئة الأصلية. لنقم بإنشاء مثيل جديد باستخدام فئة `Knight` .

In [26]:
```
my_knight = Knight()
```

In [27]:
```
my_knight.move()
```

```
You can move.
```

يمكنك أن ترى أن المثيل الذي تم تحديده استنادًا إلى الفئة الفرعية يرث ويستخدم `move` المحدد في الفئة الرئيسية `person` .

(2) Overriding

عندما يتم تعريف طريقة في فئة فرعية بنفس الاسم كما في الفئة الأصلية ، سيتم تجاوز ذلك من الفئة الأصلية بواسطة وظيفة الطريقة الجديدة في الفئة الفرعية.

In [28]:
```
# Parent class
class Person:
    def move(self):
        print("You can move.")
# Child class
class Archer(Person):
    def move(self):
        print("You can climb a tree.")
```

In [29]:
```
my_archer = Archer()
my_archer.move()
```

```
You can climb a tree.
```

[ممارسة] وراثة الفئة

بناءً على ما تعلمته ، اكتب إجابات الأسئلة في خلية الكود أدناه.
وراثة فئة الشركة التي قمت بإنشائها من قبل ، قم بإنشاء فئة خاصة للشركة.

تمرين 6.أنشئ فئة `company` بالمتطلبات أدناه. (نصيحة: راجع قسم الممارسة السابق.)

- طريقة منشئ تخزن `name` , `id` و `owner` .
- `print_info` أسلوب الذي يطبع به معلومات الشركة(`name` , `id` و `owner`)

In [30]:
```
# Exercise6 - Answer
class Company():
    def __init__(self, id, name, owner):
        self.id = id
        self.name = name
        self.owner = owner

    def print_info(self):
        print(f"ID : {self.id}")
        print(f"Name : {self.name}")
        print(f"Owner : {self.owner}")
```

تمرين 7. عند وراثة فئة `company` ، قم بإنشاء فئة باسم `BondedWarehouseCompany` . أنشئ طريقة `store_product` التي تُخرج `store_product` . داخل فئة `BondedWarehouseCompany` .

In [31]:
```
# Exercise7 - Answer
class BondedWarehouseCompany(Company):
    def store_product(self):
        print("Storing freight.")
```

تمرين 8. عند وراثة فئة `company` ، أنشئ فئة باسم `TransitCompany` . إنشاء طريقة `transport_product` التي تنتج `transporting freight` . داخل فئة `TransitCompany` .

In [32]:
```
# Exercise8 - Answer
class TransitCompany(Company):
    def transport_product(self):
        print("Transporting freight.")
```

تمرين 9. قم بإنشاء موجود شركة باستخدام فئة `BondedWarehouseCompany` بالمعلومات أدناه ، ثم قم بتعيينه إلى المتغير `bonded_warehouse_company` .

- `name` : bonded_warehouse_company
- `owner` : Jake
- `TIN` : 1153

In [33]:
```
# Exercise9 - Answer
bonded_warehouse_company = BondedWarehouseCompany(1153, 'bonded_warehouse_company
```

تمرين 10.قم بإنشاء موجود شركة باستخدام فئة `TransitCompany` بالمعلومات الواردة أدناه ، ثم قم بتعيينها إلى المتغير `transit_company`

- `name` : transit_company
- `owner` : Jane
- `TIN` : 3215

In [34]:
```python
# Exercise10 - Answer
transit_company = TransitCompany(3215, 'transit_company', 'Jane')
```

تمرين11. قم بإخراج **id** و **name,** و **owner** من **bonded_warehouse_company** و **transit_company**

In [35]:
```python
# Exercise11 - Answer
print(bonded_warehouse_company.id)
print(bonded_warehouse_company.name)
print(bonded_warehouse_company.owner)

print(transit_company.id)
print(transit_company.name)
print(transit_company.owner)
```

```
1153
bonded_warehouse_company
Jake
3215
transit_company
Jane
```

تمرين12.قم بتنفيذ طريقة **print_info** وطريقة **store_product** من **bonded_warehouse_company.**

In [36]:
```python
# Exercise12 - Answer
bonded_warehouse_company.print_info()
bonded_warehouse_company.store_product()
```

```
ID : 1153
Name : bonded_warehouse_company
Owner : Jake
Storing freight.
```

تمرين13. قم بتنفيذ أسلوب **print_info** وأسلوب **transport_product** من **transit_company.**

In [37]:
```python
# Exercise13 - Answer
transit_company.print_info()
transit_company.transport_product()
```

```
ID : 3215
Name : transit_company
Owner : Jane
Transporting freight.
```